PHILIPS

PHILIPS

MSX-BASIC

MSX™

ESTE MANUAL CONTIENE INFORMACIÓN DE TODAS LAS SENTENCIAS MSX-BASIC

A. SICKLER

New Media Systems

MSX-BASIC

Edición de Philips

Albert Sickler

MSX-BASIC

Edición de Philips

Kluwer Technische Boeken B.V.
Deventer – Antwerpen

1a edición 1986
2a edición 1986
3a edición 1987
4a edición 1987

PROLOGO

Detrás de las siglas MSX se encuentra escondido todo un mundo, un mundo que está determinado por una gran cantidad de marcas diferentes de ordenadores que, extrañamente, llaman la atención más por sus similitudes que por sus diferencias. Ahora, por primera vez en el mundo de los ordenadores, se introduce una norma cuyas consecuencias serán, sin duda, enormes.

Con la introducción de los ordenadores MSX para el gran público, se desencadenará, sin lugar a dudas, un enorme interés en torno a los ordenadores MSX y qué es lo que puede hacerse con el MSX-BASIC

Usted, al decidirse por un ordenador MSX, ha escogido, finalmente, un ordenador Philips. Esta es una buena elección, pues todo el mundo sabe cómo se ha ganado este campeón de la electrónica los trofeos en el campo de la calidad.

Para muchos, ésta será una primera toma de contacto. En ella, el lector confía, sin duda, encontrar la comprensión que necesita por las dificultades que tendrá que superar. Para estos lectores ha sido escrito este manual.

CONTENIDO

INTRODUCCIÓN AL BASIC

1

LOS PRIMEROS PASOS: SOBRE PRINT Y EL PROCESAMIENTO DE CALCULOS ARITMETICOS

PRINT para la ejecución de cálculos aritméticos directos

Al encender el ordenador, éste se coloca automáticamente en la disposición de poder trabajar con el BASIC.
Para demostrarlo, escribimos:

```
PRINT 2+3
```

En la pantalla aparecerá, al mismo tiempo, PRINT 2+3. Sólo al pulsar la tecla RETURN, cumplirá la orden el ordenador. La tecla RETURN se indica, tanto con la palabra RETURN, como con el símbolo ↵.

PRINT quiere decir 'imprima' o, mejor, 'exponga', y después de pulsar la tecla RETURN, vemos que algo ha sido expuesto, es decir:

```
5
```

y el término:

```
Ok
```

Está claro que 5 es la respuesa al problema formulado 2+3. El término Ok indica que el ordenador ha terminado su tarea y que podemos imponerle otra.
Observaciones:

- En el ordenador no nos está permitido escribir 2+3= (y después, por ejemplo, pulsar la tecla RETURN). Si hacemos esto, aparecerá el aviso -en inglés, desde luego - de que hemos hecho algo mal.
- En el ordenador una orden, cuyo resultado queremos ver inmediatamente en la pantalla, tiene que empezar siempre con el término PRINT.

Multiplicación y división:

Para la multiplicación se utiliza el símbolo * y, para la división la rayita oblicua de la división. Así, la instrucción PRINT 5*36 da como resultado 180, y PRINT 180/36 da como resultado 5.

Paréntesis

Los paréntesis los podemos utilizar como aprendimos en la escuela; así

$$\frac{5+5}{23(17+103)}$$

se escribe como sigue:

```
PRINT (5+15)/(23*(17+103))
```

En primer lugar hay que tener en cuenta que la expresión completa se ha colocado en un solo renglón. El numerador y el denominador se han colocado ahora entre paréntesis. Si no hubiésemos hecho ésto, la instrucción PRINT sería:

```
PRINT 5+15/23*(17+103)
```

Esta instrucción conduce a un resultado distinto del deseado (esto se entenderá mejor a continuación, cuando expliquemos las reglas de prioridad en el cálculo).

Además vemos que en el denominador, entre 23 y la apertura del paréntesis se ha colocado un signo de multiplicar. De lo que se trata en la expresión original del denominador 23(17+103) es de multiplicar 23 por (17+103).

Orden de ejecución de las operaciones

En la escuela aprendimos reglas nemotécnicas para recordar que multiplicar iba antes que dividir, o dicho más formalmente, multiplicar tiene prioridad sobre dividir. En el BASIC se tienen en cuenta las siguientes prioridades:

1. Primero se solucionan las expresiones entre paréntesis;
2. Multiplicar y dividir tienen la misma prioridad y van antes que sumar y restar, que también tienen la misma prioridad;
3. Las operaciones de la misma prioridad se solucionan de izquierda a derecha.

Por ejemplo:

```
PRINT              15/3*5

da                 5*5

y, por lo tanto,   25
```

Luego el resultado es 25 (¡y no 1!)

En el Apéndice F se encuentra un cuadro de la totalidad de las reglas de prioridad.

Texto

El texto a presentar se coloca siempre entre comillas. Ejemplo:

```
PRINT "ESTE ES UN EJEMPLO"
```

da como resultado:

```
ESTE ES UN EJEMPLO
```

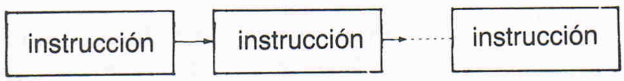
Observemos que las comillas no aparecen en el resultado. Más adelante estudiaremos otras posibilidades.

2

¡Y AHORA UN PROGRAMA BASIC!

Un programa BASIC: uso de los números de linéa

Como se ha observado en el capítulo 1, un programa no es más que una sucesión de instrucciones que cumple el ordenador automáticamente, después de que se le haya dado cierto comando para ello.
Esquemáticamente:



En el caso más sencillo podemos escribir el programa en el teclado, como si se tratara de un trozo de texto. Esto implica que este programa, o sea, esa sucesión de instrucciones, tienen que ser guardadas primero en la memoria.

Pero ¿cómo conseguiremos guardar las instrucciones que componen el programa en la memoria del ordenador antes de que sean cumplidas?
La respuesta es sencilla:

poniéndoles números delante a las instrucciones.

A continuación ilustraremos esto con ayuda de ejemplos.

Un ejemplo:

Supongamos que queremos que el ordenador, como programa, ejecute las tres instrucciones siguientes consecutivamente:

```
PRINT "ESTE ES"
PRINT "EL PRIMER EJEMPLO"
PRINT "DE PROGRAMMA"
```

entonces, al escribir este programa - al fin y al al cabo no se trata más que de una *sucesión* de instrucciones - tenemos que poner un número delante de cada instrucción.
Normalmente numeraremos con 10, 20, 30, etc.

Así pues, escribimos:

```
10 PRINT "ESTE ES"
20 PRINT "EL PRIMER EJEMPLO"
30 PRINT "DE PROGRAMA"
```

Al final de cada renglón pulsamos de la tecla RETURN para indicar que se ha acabado de escribir el renglón (¡No hay que olvidarse de pulsar RETURN también después del último renglón!).

Cuando se haya terminado de escribir todos los renglones, el programa completo se hallará almacenado en la memoria. Y, además, lo estamos viendo en la pantalla.

¡Vamos a ejecutar el programa!

Si escribimos:

```
RUN
```
(seguido de la palabra RETURN)

en la pantalla aparecerá:

```
ESTE ES
EL PRIMER EJEMPLO
DE PROGRAMA
```

¡Bravo! hemos introducido un programa en el ordenador.

En este caso el programa se componía sólo de instrucciones PRINT y el texto a exponer; pero no por eso deja de ser una secuencia de instrucciones, o sea, un programa.

Junto a esto, haremos las siguientes observaciones:

- Después de haber sido ejecutado el programa, éste se queda almacenado en la memoria. Sólo si damos la orden de borrarlo, o si desconectamos el ordenador de la corriente eléctrica, se perderá el programa.
- Los números indican además el orden en que se cumplen las instrucciones, yendo del más bajo al más alto. Incluso habríamos podido escribir:

```
10 PRINT "ESTE ES"
30 PRINT "DE PROGRAMA"
20 PRINT "EL PRIMER EJEMPLO"
```

El resultado habría sido el mismo, ya que los números de los renglones así lo indican. El ordenador colocará la instrucciones en el orden correcto inmediatamente después de que se hayan escrito.

- Normalmente hablaremos de número de línea y no de números. Como números de línea no podemos utilizar más que los números enteros (de 0 a 65529).
- El hecho de numerar las líneas de programa con 10, 20, 30, etc. tiene la ventaja de, si se quiere, poder añadir más tarde instrucciones entre las ya existentes.
- RUN es un comando; o sea, que el ordenador debe ejecutar una acción determinada inmediatamente después de que se le huya dado esta instrucción.

LIST para la exposición del programa. Teclea LIST seguido de la tecla RETURN.

AUTO para la creación automática de números de líneas.

RENUM para la renumeración de los números de línea.

DELETE para borrar partes del programa.

NEW para borrar la memoria.

Estas órdenes se describen más detalladamente en el Resumen de las Instrucciones del MSX-BASIC. Ensaya cada una de ellas.

Añadir, insertar y modificar líneas

En la práctica ocurre a memudo que tendremos que modificar un programa. Ilustramos esto con la ayuda del siguiente programa:

```
10 PRINT "ESTO ES UN"
20 PRINT "PROGRAMA"
30 PRINT "COMO ILUSTRACION"
```

Tecleemos ahora el programa sin olvidarnos de borrar eventualmente el programa anterior con NEW.

Añadir

Añadir líneas pasa más o menos desapercibido, escojamos un número de línea superior al último, por ejemplo:

```
40 PRINT "ACERCA DEL MANEJO"
50 PRINT "DE LOS NUMEROS DE LINEA"
```

Ensayemos la instrucción LIST para convencernos de que estas líneas realmente han sido añadidas.

Inserción

Cuando queramos insertar una línea, se escoge un número de línea que esté entre dos líneas entre las cuales queremos situar la nuestra. Por ejemplo:

```
15 PRINT "SENCILLO"
```

Controlemos con LIST si de verdad esta línea se encuentra en su lugar correcto.

Borrar

Se puede sencillamente borrar una línea si se teclea el número de la misma y después la tecla RETURN, por ejemplo:

```
15 (tecla RETURN)
```

Controlemos nuevamente el resultado con la instrucción LIST. También se puede opcionalmente utilizar el comando DELETE (Véase el Resumen de las instrucciones MSX-BASIC).

Corrección de errores

En lo que llevamos dicho, de hecho, hemos tratado ya de algunas formas de cambiar un programa. En este párrafo mostraremos qué más se puede hacer si se comete un error.

La solución más sencilla, a primera vista, parece ser el volver a escribir la línea errónea, pero esta vez sin faltas.
Escribamos:

```
20 PRINT "SINPLEMENTE"
```

Después del comando LIST, vemos que esta línea ha sido incluida en el programa con un error garrafal. Podemos corregir la falta volviendo a escribir la línea correctamente, desde luego. Pero más elegante será hacer funcionar las teclas del cursor. Estas son unas teclas con flechas que se encuentran a la derecha del

teclado. Si se pulsan esas teclas, se verá cómo se traslada de sitio en la pantalla el cuadradito blanco.

Este cuadradito es el cursor y de ahí que las teclas de las flechas se llamen teclas de control del cursor.

Movamos pues el cursor, de tal manera, que se coloque exactamente encima del error, o sea:

```
20 PRINT "SI[N]PLEMENTE"
            ↑
```

el cursor señala

Ahora pulsemos la tecla M y directamente RETURN. A continuación, volvamos a desplacar el cursor hasta que quede debajo del programa y...¡eureka, está corregido!

Además, también podemos mantener una de las teclas del cursor pulsada, con lo que se produce el mismo efecto que si se pulsara repetidamente. Este efecto de 'autorrepetición' se consigue también en todas las demás teclas.

Teclas especiales

Finalmente mencionaremos que, mientras que se está escribiendo, también se puede corregir el texto pulsando la tecla BS. Con la tecla INS podemos insertar signos y con la DEL podemos borrarlos. Estas dos últimas teclas son de importancia si se trabaja con las teclas del cursor. Primero se traslada el cursor hasta el lugar en que se ha constatado para después, cambiar, borrar o añadir signos, a discreción.

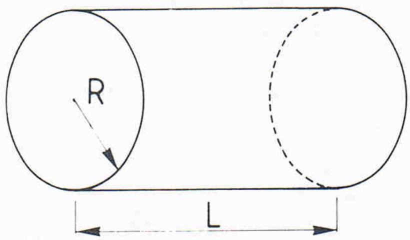
3 LAS VARIABLES Y ALGO MÁS SOBRE CALCULOS

Introducción

Empezaremos con algunos preliminares.

Para calcular el volumen de un cilindro, tenemos que saber la superficie de la sección y multiplicar el valor de ésta por la altura.

Véase gráficamente:



superficie de la sección = 3,14159×R×R

Volumen del cilindro = superficie de la sección × L = 3,14159×R×R×L

Vemos que el volumen del cilindro depende, en general, de dos magnitudes: el radio R y la longitud L.

El programa siguiente calculará el volumen de un cilindro, con los valores:

L=5 y
R=7

Programa:

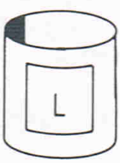
```
10 L=5
20 R=7
30 PI=3.14159
40 SUP=PI*R*R
50 V=SUP*L
60 PRINT V
```

Después del comando RUN aparece:

```
769.68955
```

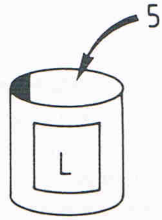
Veamos ahora el programa, línea por línea: En la línea 10 vemos la expresión:

L=5



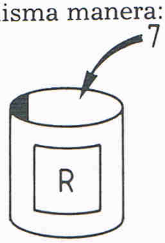
El efecto de esta expresión es que el ordenador reserva un trozo de memoria pegándole, imaginariamente, la etiqueta L. Este trozo de memoria podemos imaginárnoslo como si fuera una caja, una 'gaveta'.

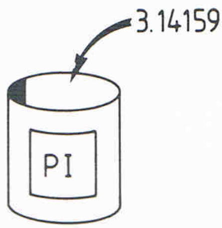
La expresión L=5 quiere decir, pues, que en la gaveta L está guardado el número 5.
Gráficamente:



Si en el programa se menciona L (línea 50), el ordenador lo que hace, en realidad, es mirar cuál es el contenido de esa gaveta, y usarlo.

Las líneas 20 y 30 podemos representarlas gráficamente de la misma manera:





Observe que en PI se guarda el número 3.14159 y no 3,14159. O sea que la coma que utilizamos nosotros, normalmente, en el BASIC se representa por un punto. En el resultado del programa vemos también este 'punto decimal'.

Quien tenga algunos conocimientos de matemáticas, habrá reconocido en el número 3.14159 el número π (pronunciación: pi).

La línea 40 muestra una fórmula: en la caja con la etiqueta SUP será guardado el valor que se obtiene al multiplicar PI por R×R.
Luego en SUP se guardará el valor 3.14159×7×7.

Este valor que acabamos de hallar vuelve a ser utilizado en la línea 50 para determinar el valor del volumen. Este será guardado en la gaveta de la etiqueta V.

Finalmente, utilizamos una instrucción PRINT para exponer el contenido de este último compartimento reservado de la memoria:

```
PRINT V
```

↗ representa — ↖ el contenido de la caja con la etiqueta V

Observemos que aquí no se expone la letra V pues, para eso, la instrucción tendría que haber sido PRINT 'V', o sea que teníamos que haber puesto la V entre las famosas comillas.

A propósito de este primer programa haremos las observaciones siguientes:

- En lugar de

```
10 L=5
```

podíamos haber escrito

```
10 LET L=5
```

Que significaría, literalmente, 'Hagamos que L sea igual a 5' o mejor, 'Depositemos en la gaveta L el número 5'.

Con esta última expresión queremos acentuar el hecho de que el ordenador ejecuta una acción: *mete* algo en la gaveta que tiene la etiqueta L. Esta acción quedará completamente clara con el siguiente ejemplo:

```
10 LET X=3
20 LET X=X+4
```

En la segunda línea se le da a X el antiguo valor de X al que se le ha sumado 4, es decir 3+4. Así, X tendrá al final el valor de 7. Es de notar que una expresión como X=X+4 es absolutamente correcta en BASIC mientras que matemáticamente no está permitida.

- El hecho de que podamos encerrar un valor arbitrario en una de esas gavetas indica que su contenido puede ser variable. Esta es la razón por la cual hablamos de una *variable* en lugar de hablar de gavetas. Nuestro programa utiliza las variables L, R, PI, SUP y V.
A partir de ahora hablaremos también de la 'asignación de valores a una variable' en lugar de hablar de 'meter un valor en una gaveta'.

- Para dar nombres a las variables existen unas reglas determinadas. Estas son:
  - Sólo las dos primeras letras de un nombre son reconocidas por el ordenador como el verdadero nombre. Así, los nombres UNO y UNICO son iguales para el ordenador: sólo es reconocido UN.
  - No pueden usarse como nombres palabras reservadas. Así por ejemplo, no puede usarse IF como nombre, ya que este término está reservado.

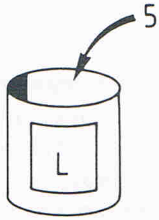
  Los nombres reservados pueden verse en el apéndice.

Funciones estándar

El ejemplo presentado era aún muy sencillo. No podemos imaginarnos que de esta manera podamos ejecutar cálculos complicados. Al tratar las llamadas funciones estándar aumentará notablemente el conocimiento de las posibilidades de cálculo que incluye el BASIC.

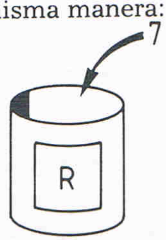
Para explicar las funciones estándar, volvemos a hacer una escapada a la calculadora, y concretamente, a la calculadora con funciones matemáticas. Si en una calculadora tenemos que cal-

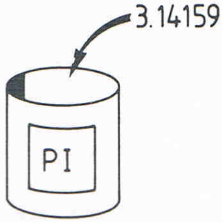
La expresión L=5 quiere decir, pues, que en la gaveta L está guardado el número 5.
Gráficamente:



Si en el programa se menciona L (línea 50), el ordenador lo que hace, en realidad, es mirar cuál es el contenido de esa gaveta, y usarlo.

Las líneas 20 y 30 podemos representarlas gráficamente de la misma manera:





Observe que en PI se guarda el número 3.14159 y no 3,14159. O sea que la coma que utilizamos nosotros, normalmente, en el BASIC se representa por un punto. En el resultado del programa vemos también este 'punto decimal'.

Quien tenga algunos conocimientos de matemáticas, habrá reconocido en el número 3.14159 el número π (pronunciación: pi).

La línea 40 muestra una fórmula: en la caja con la etiqueta SUP será guardado el valor que se obtiene al multiplicar PI por R×R.
Luego en SUP se guardará el valor 3.14159×7×7.

Este valor que acabamos de hallar vuelve a ser utilizado en la línea 50 para determinar el valor del volumen. Este será guardado en la gaveta de la etiqueta V.

Finalmente, utilizamos una instrucción PRINT para exponer el contenido de este último compartimento reservado de la memoria:

```
PRINT V
```

↗ representa — ↖ el contenido de la caja con la etiqueta V

Observemos que aquí no se expone la letra V pues, para eso, la instrucción tendría que haber sido PRINT 'V', o sea que teníamos que haber puesto la V entre las famosas comillas.

A propósito de este primer programa haremos las observaciones siguientes:

- En lugar de

```
10 L=5
```

podíamos haber escrito

```
10 LET L=5
```

Que significaría, literalmente, 'Hagamos que L sea igual a 5' o mejor, 'Depositemos en la gaveta L el número 5'.

Con esta última expresión queremos acentuar el hecho de que el ordenador ejecuta una acción: *mete* algo en la gaveta que tiene la etiqueta L. Esta acción quedará completamente clara con el siguiente ejemplo:

```
10 LET X=3
20 LET X=X+4
```

En la segunda línea se le da a X el antiguo valor de X al que se le ha sumado 4, es decir 3+4. Así, X tendrá al final el valor de 7. Es de notar que una expresión como X=X+4 es absolutamente correcta en BASIC mientras que matemáticamente no está permitida.

- El hecho de que podamos encerrar un valor arbitrario en una de esas gavetas indica que su contenido puede ser variable. Esta es la razón por la cual hablamos de una *variable* en lugar de hablar de gavetas. Nuestro programa utiliza las variables L, R, PI, SUP y V.

  A partir de ahora hablaremos también de la 'asignación de valores a una variable' en lugar de hablar de 'meter un valor en una gaveta'.

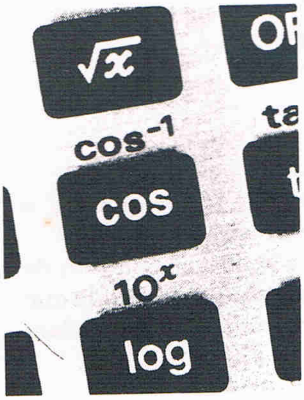
- Para dar nombres a las variables existen unas reglas determinadas. Estas son:
  - Sólo las dos primeras letras de un nombre son reconocidas por el ordenador como el verdadero nombre. Así, los nombres UNO y UNICO son iguales para el ordenador: sólo es reconocido UN.
  - No pueden usarse como nombres palabras reservadas. Así por ejemplo, no puede usarse IF como nombre, ya que este término está reservado.

  Los nombres reservados pueden verse en el apéndice.

Funciones estándar

El ejemplo presentado era aún muy sencillo. No podemos imaginarnos que de esta manera podamos ejecutar cálculos complicados. Al tratar las llamadas funciones estándar aumentará notablemente el conocimiento de las posibilidades de cálculo que incluye el BASIC.

Para explicar las funciones estándar, volvemos a hacer una escapada a la calculadora, y concretamente, a la calculadora con funciones matemáticas. Si en una calculadora tenemos que cal-



cular la raíz de un número, introducimos un número y luego pulsamos la tecla que tenga encima el símbolo de raíz. Así se calcula, pulsando simplemente la tecla adecuada, la función que se desee.

En el BASIC podemos calcular los valores de las funciones de una manera igual de sencilla. Con un nombre determinado llamaremos a cada función de ésas, poniéndole entre paréntesis el número del cual hay que calcular el valor de la función correspondiente.

Ejemplos:

```
10 A=SQR(4)
20 PRINT A
```

Resultado:

```
2
```

En la línea 10 se asigna como valor de A la raíz cuadrada de 4. El nombre clave que designa la función, en este ejemplo, es SQR. Entre paréntesis se encuentra el valor del cual debe sacarse la raíz, en este caso (para no dificultar las cosas) 4. A ese valor sobre el que se aplica la función, se le llama argumento.

Tenemos que observar que entre los paréntesis también puede ponerse una variable o una expresión aritmética.

Ejemplos:

`B=SQR(A)`	: Aquí se expresa el argumento de la función por medio de la variable A.
`C=SQR(SQR(5)+4+A)`	: Aquí se expresa el argumento de la función por medio de la expresión SQR(5)+4+A. Oservemos que esta expresión contiene, a su vez, otra función.

En el resumen de las instrucciones MSX-BASIC se encontrarà un cuadro con todas las instrucciones posibles.

Designación MSX-BASIC	*Notación algebraica*	*Significado*
ABS (X)	$\|x\|$	Da el valor absoluto de X. Este es el valor 'sin signo'.
ATN (X)	arctan(x)	Da el arcotangente de X. Solución entre -pi/2 y pi/2.
COS (X)	cos(x)	Da el coseno de X (X en radianes).
EXP (X)	e^x	Da la potencia 'e' elevada a X.
INT (X)	ninguna	Redondea el valor de X siempre hacia abajo. Así, INT (3.8) da el valor 3 y INT (-3.1) el valor -4. Podemos redondear 'verdaderamente' un número con INT (X+0.5).
LOG (X)	${}^{e}\log(x)$	Da el llamado logaritmo natural de X.
SNG (X)	ninguna	Da el valor -1.0, ó 1, dependiendo del hecho de si X es negativo, cero, o positivo.
SIN (X)	sin(x)	Da el seno de X (X en radianes).
SQR (X)	$\sqrt{x}$	Da la raíz cuadrada de X.
TAN (X)	tan(x)	Da la tangente de X (X en radianes).

Queremos hacer notar que en esta tabla no aparece la elevación a potencias. Para esto utilizamos el signo ∧. Así, PRINT 2 ∧ 3 da el valor 8 como resultado ($=2^3$).

Varias instrucciones en una línea

El MSX-BASIC nos ofrece la posibilidad de situar varias instrucciones en la misma línea.

Entonces, tenemos que utilizar el símbolo: como signo de separación. El siguinte programa ilustrará esto.

sin signo de separación

```
10 A=10
20 B=5
30 C=A+B
40 PRINT C
```

con signos de separación

```
10 A=10:B=5:C=A+B:PRINT C
```

4 INPUT, READ Y DATA

Input La primera de las instrucciones que vamos a tratar ahora es INPUT. Esta nos ofrece la posibilidad de poner valores 'dentro' de las variables durante la ejecución de un programa. Pero más importante todavía es, quizás, el hecho de que esta instrucción nos ofrece la posibilidad de confeccionar programas de uso general. Fijémonos de nuevo en el programa del capítulo anterior:

```
10 L=5
20 R=7
30 PI=3.14159
40 SUP=PI*R*R
50 V=SUP*L
60 PRINT V
```

En lugar de este programa pudimos haber escrito, naturalmente:

```
PRINT 3.14159*7*7*5
```

para obtener el mismo resultado.

No lo hemos hecho porque queríamos mostrar cómo podemos almacenar un cálculo aritmético en la memoria del ordenador; y en el cual, para determinar el resultado, sólo es necesario rellenar los valores de R y L.

Para determinar, por ejemplo, el volumen de un cilindro, siendo L=34 y R=10, podemos modificar las lineas 10 y 20 a:

```
10 L=34
20 R=10
```

y después del comando RUN aparecerá la respuesta deseada. Pero la modificación de un programa para la ejecución de operaciones aritméticas no es muy aconsejable, es fácil cometer errores...

Sin embargo el programa quedaría corregido si después del comando RUN apareciese en la pantalla un mensaje diciendo que deben ser ingresados los valores para L y R respectivamente.

Ahora bien, esta posibilidad existe en la forma de la instrucción INPUT que es como sigue:
INPUT 'texto a exponer'; nombre de la variable
Por ejemplo:

```
INPUT "INGRESAR LONGITUD"; L
```

El resultado de esta instrucción es que el ordenador expone el texto seguido de un signo de interrogación e interrumpe el programa inmediatamente después. El usuario debe inscribir ahora un número mediante el teclado, por ejemplo:

5 (más tecla RETURN)

Inmediatamente después de pulsar la tecla RETURN se le asigna este número a la variable L y el ordenador continuará con la ejecución del programa.

Después de las instrucciones INPUT nuestro programa quedará así:

```
10 INPUT "TECLEA LONGITUD"; L
20 INPUT "TECLEA RADIO"; R
30 PI=3.14159
40 SUP=PI*R*R
50 V=SUP*L
60 PRINT "EL VOLUMEN="; V
```

El resultado podría ser:

```
TECLEA LONGITUD? 5
TECLEA RADIO? 7
EL VOLUMEN= 769.68955
```

Aquí vemos cómo con la instrucción INPUT ha sido incluida en este programa y también, un texto; colocándose dicho texto entre comillas y añadiéndosele detrás el signo; como signo de separación.

El programa ha quedado así, mucho mejor que nuestro programa original. La estructura ha quedado fijada. Cuando queramos calcular el volumen de un cilindro *cualquiera*, no necesitaremos hacer modificaciones en el propio programa.

Observaciones

A propósito de la instrucción INPUT añadiremos las siguientes observaciones:

- En una instrucción INPUT está permitido incluir más de una variable, por ejemplo:

```
INPUT "INGRESAR A, B, Y C"; A, B, C
```

Después de la interrupción del programa, el ordenador muestra el texto y el signo de interrogación y luego tenemos que teclear tres números separados por comas.

- En la instrucción INPUT podemos, también, omitir el texto, por ejemplo:

```
INPUT A
```

Después de la interrupción, el ordenador muestra, ahora, sólo el signo de interrogación, como señal de que tenemos que ingresar algo por teclado.

READ y DATA

La tercera posibilidad, de la que ahora vamos a hablar, se relaciona con el deseo de darle valores a una serie grande de variables. Consideramos aquí entonces que estos valores, común-

4 INPUT, READ Y DATA

Input La primera de las instrucciones que vamos a tratar ahora es INPUT. Esta nos ofrece la posibilidad de poner valores 'dentro' de las variables durante la ejecución de un programa. Pero más importante todavía es, quizás, el hecho de que esta instrucción nos ofrece la posibilidad de confeccionar programas de uso general. Fijémonos de nuevo en el programa del capítulo anterior:

```
10 L=5
20 R=7
30 PI=3.14159
40 SUP=PI*R*R
50 V=SUP*L
60 PRINT V
```

En lugar de este programa pudimos haber escrito, naturalmente:

```
PRINT 3.14159*7*7*5
```

para obtener el mismo resultado.

No lo hemos hecho porque queríamos mostrar cómo podemos almacenar un cálculo aritmético en la memoria del ordenador; y en el cual, para determinar el resultado, sólo es necesario rellenar los valores de R y L.

Para determinar, por ejemplo, el volumen de un cilindro, siendo L=34 y R=10, podemos modificar las lineas 10 y 20 a:

```
10 L=34
20 R=10
```

y después del comando RUN aparecerá la respuesta deseada. Pero la modificación de un programa para la ejecución de operaciones aritméticas no es muy aconsejable, es fácil cometer errores...

Sin embargo el programa quedaría corregido si después del comando RUN apareciese en la pantalla un mensaje diciendo que deben ser ingresados los valores para L y R respectivamente.

Ahora bien, esta posibilidad existe en la forma de la instrucción INPUT que es como sigue:
INPUT 'texto a exponer'; nombre de la variable
Por ejemplo:

```
INPUT "INGRESAR LONGITUD"; L
```

El resultado de esta instrucción es que el ordenador expone el texto seguido de un signo de interrogación e interrumpe el programa inmediatamente después. El usuario debe inscribir ahora un número mediante el teclado, por ejemplo:

5 (más tecla RETURN)

Inmediatamente después de pulsar la tecla RETURN se le asigna este número a la variable L y el ordenador continuará con la ejecución del programa.

Después de las instrucciones INPUT nuestro programa quedará así:

```
10 INPUT "TECLEA LONGITUD"; L
20 INPUT "TECLEA RADIO"; R
30 PI=3.14159
40 SUP=PI*R*R
50 V=SUP*L
60 PRINT "EL VOLUMEN="; V
```

El resultado podría ser:

```
TECLEA LONGITUD? 5
TECLEA RADIO? 7
EL VOLUMEN= 769.68955
```

Aquí vemos cómo con la instrucción INPUT ha sido incluida en este programa y también, un texto; colocándose dicho texto entre comillas y añadiéndosele detrás el signo; como signo de separación.

El programa ha quedado así, mucho mejor que nuestro programa original. La estructura ha quedado fijada. Cuando queramos calcular el volumen de un cilindro *cualquiera*, no necesitaremos hacer modificaciones en el propio programa.

Observaciones

A propósito de la instrucción INPUT añadiremos las siguientes observaciones:

- En una instrucción INPUT está permitido incluir más de una variable, por ejemplo:

```
INPUT "INGRESAR A, B, Y C"; A, B, C
```

Después de la interrupción del programa, el ordenador muestra el texto y el signo de interrogación y luego tenemos que teclear tres números separados por comas.

- En la instrucción INPUT podemos, también, omitir el texto, por ejemplo:

```
INPUT A
```

Después de la interrupción, el ordenador muestra, ahora, sólo el signo de interrogación, como señal de que tenemos que ingresar algo por teclado.

READ y DATA

La tercera posibilidad, de la que ahora vamos a hablar, se relaciona con el deseo de darle valores a una serie grande de variables. Consideramos aquí entonces que estos valores, común-

mente, de programa a programa, no sufrirán ningún cambio.

En una situación así, podríamos naturalmente insertar toda una serie de instrucciones LET, pero, como veremos, el procedimiento es mucho más elegante utilizando READ y DATA. Mostraremos enseguida un ejemplo:

```
10 READ A, B, C, D
20 F=A+B+C+D
30 PRINT F
40 DATA 3, 7, 4, 8
```

La consecuencia del empleo de la instrucción READ en la línea 10, es que el ordenador sabe que se van a mencionar esas variables, cuyos valores estarán reseñados en una linea que comience por DATA.

Así se le asigna a A el valor de 3, a B el valor de 7, a C el valor de 4 y a D el valor de 8.
El resultado del programa confirma esto:

```
22
```

Observe que la línea de DATA no constituye, en sí, una instrucción ejecutable. Es una especie de 'apuntador' donde el ordenador encuentra los valores que necesita. Por otra parte, podíamos haber dividido en dos la lista de variables usada con READ:

```
10 READ A, B
15 READ C, D
```

y, del mismo modo, habríamos podido dividir la lista de valores constantes reseñados en DATA, por ejemplo, así:

```
40 DATA 3, 7, 4
50 DATA 8
```

Al ordenador todo esto le da lo mismo; él toma esos datos como si formaran una lista continua. Uno puede comprender mejor el mecanismo imaginando que es, como si el ordenador colocara de antemano un puntero dirigido hacia el primer valor de la lista de DATA. Cada vez que se toma un valor para asignarlo a una variable por medio de la instrucción READ, se traslada ese puntero un lugar para apuntar hacia el siguiente valor de la lista.

```
40 DATA 3, 7, 4, 8
           ↑
```

el puntero corre cada vez un lugar

Cuando ha llegado al último valor de la lista de DATA, el ordenador corre su puntero al primer valor de la lista de DATA siguiente, si es que la hay.

También podemos hacer que vaya ese puntero a su posición original, con ayuda de la siguiente instrucción:

```
RESTORE
```

Ejemplo:

```
10 READ A, B
20 RESTORE
30 READ C, D
40 F=A+B+C+D
50 PRINT F
60 DATA 3, 7, 4, 8
```

Resultado:

```
20
```

Observemos que el resultado ahora es 20. Esto se deriva de la instrucción
RESTORE: se ha asignado 3 y 7 tanto a A y B como a C y D.

5 NUMEROS, GRANDES, PEQUEÑOS Y DE TODAS LAS FORMAS

Introducción

En relacion con lo siguiente observemos primero la tabla de abajo.

De la tabla podemos deducir, por ejemplo, que 10^4 corresponde a 10000, o sea, la cantidad de ceros coincide con el exponente (numerito que hay a la derecha, encima del 10).

En palabras	*notación*	*significado*	*valor*
10 elevado a 1	10^1	10	10
10 elevado a 2	10^2	10×10	100
10 elevado a 3	10^3	10×10×10	1000
10 elevado a 4	10^4	10×10×10×10	10000
10 elevado a -1	10^{-1}	1/10	0.1
10 elevado a -2	10^{-2}	1/(10×10)	0.01
10 elevado a -3	10^{-3}	1/(10×10×10)	0.001
10 elevado a 0	10^0	10/10	1

Ejemplo
¿Con qué potencia se corresponde el 1000 000 000?
Respuesta: Cuente el número de ceros. Son 9, así que es 10^9.

De la tabla se desprende asimismo, que 0.001 se corresponde con 10^{-3}.

Así pues, también con números menores de 1, puede por lo visto, contarse la cantidad de ceros para determinar la potencia.

Ejemplo
¿A qué potencia corresponde el número siguiente?

0.000 000 000 000 000 000 1

Respuesta: Cuente los ceros. Son 19, así que es 10^{-19}.

También podemos escribir esto así: 1×10^{-19}

el primer número se denomina *mantisa* y el segundo se designa como *exponente.*

En el ordenador, el número anterior se expresa en la siguiente notación:

```
1E-19
```

En otras palabras, utilizamos la letra E para diferenciar la mantisa del exponente.

Para adquirir un poco de práctica en esta notación, puede utilizarse el programa siguiente:

```
10 INPUT A
20 INPUT B
30 C=A*B
40 PRINT C
```

Ejemplos

A=987654	B=456789	da: 451149483006
A=98765434	B=987654321	da: 9.754610765554E+16
A=.0000000008	B=.0000000007	da: 5.6E−19
A=8000000000	B=7000000000	da: 5.6E+19

Precisión doble y sencilla

Para la reproducción de números, nuestro ordenador utiliza una cantidad limitada de registros de memoria (palabras). Esto significa también que las operaciones aritméticas tienen una exactitud limitada. El MSX-BASIC ofrece al usuario la posiblidad de trabajar, tanto con la llamada precisión doble, como con la precisión sencilla. En la precisión doble se utilizan siempre 8 registros para el almacenamiento de números y en precisión sencilla solamente 4. Nuestro ordenador, de forma normal, funciona constantemente con cantidades de precisión doble. Para trabajar con las de precisión sencilla, colocaremos siempre un signo de admiración (!) detrás del número.

Las variables a las que asignemos números de precisión sencilla se diferenciarán de las variables de precisión doble por la colocación del ! inmediatamente detrás del nombre. Los ejemplos siguientes ilustrarán la diferencia entre operaciones de precisión sencilla y doble:

```
10 A=10/3
20 PRINT A
```

resultado

```
3.3333333333333
```

```
10 A!=10!/3!
20 PRINT A!
```

resultado

```
3.33333
```

A propósito de esto, observaremos todavía lo siguiente:

- Con números de precisión doble podemos colocar el signo # al final del número. Asimismo podemos utilizar esto signo como signo final de un nombre. De este modo indicamos explícitamente que estamos trabajando con números de precisión doble.
- Si con números de precisión doble utilizamos la letra D en lugar de la E (por ejemplo 23.45156321D39), indicamos explícitamente que estamos trabajando con precisión doble.
- Hay que fijarse constantemente en si merece la pena dar resultados de tantas cifras. Imaginemos que un carpintero tiene que serrar una tabla de 10 metros en 3 partes; según nuestro ordenador, cada parte sería de 3.3333333333333 m. de largo, pero ¿podrá el carpintero serrar con tanta exactitud?
- El uso de números de precisión sencilla puede ser de interés, sobre todo, si queremos ahorrar memoria. Esta problemática de

la memoria se produce en la práctica, principalmente, en el caso de las 'tablas' de datos de las que nos ocuparemos más tarde.

Números enteros

En ciertos casos queremos indicar, sin lugar a dudas, en el programa que se trata de números enteros, de los admitidos como *propios* en el ordenador. Representar esta clase de números en un programa no provoca ninguna dificultad: un número sin punto decimal es un número entero ¿no es así? Sin embargo, en las variables no se puede ver si representan un número entero o un número con punto decimal. Así pues, para hacer patente esta diferencia, se coloca el signo % detrás del nombre.
Ejemplos:

```
A% B1%
```

Ahora bien, una de esas variables sólo puede contener un entero. Si el resultado de un cálculo es décimal y este valor se le asigna a una variable-entera, este resultado automáticamente se redondea (hacia abajo).
Ejemplo:

```
10 A%=20
20 B%=3
30 C%=A%/B%
40 PRINT C%
```

Resultado:

```
6
```

Observe una vez más, que siempre se redondea hacia abajo. En este ejemplo se ha redondeado 6.6666.. en 6.

Números binarios

Para explicar los números binarios, nos fijaremos primero en cómo esta construida nuestra numeración 'normal', es decir, la numeración en el sistema de base 10. Esta está fundamentada en las potencias de 10 (véase la tabla del comienzo de este capítulo).
Ejemplo:

El número 4736 está constituido por potencias de 10, de la siguiente manera:

4 7 3 6

$6 \times 10^0 = 6 \times 1 = 6$
$3 \times 10^1 = 3 \times 10 = 30$
$7 \times 10^2 = 7 \times 100 = 700$
$4 \times 10^3 = 4 \times 1000 = 4000\ +$
4736

Aquí vemos cómo cada cifra se asocia con una potencia de 10. Partiendo de la derecha, contamos la posición cero, la uno, dos y

tres: estas posiciones coinciden exactamente con el exponente de la potencia de 10 asociada.

Pues bien, los números binarios se componen sólo de las cifras 0 y 1; y la posición relativa de cada una de estas cifras se asocia de nuevo, con una potencia, pero en este caso con una potencia de 2.

Ejemplo:

¿Con qué número decimal corresponderá el número binario 10101101? Solución

```
1   0   1   0   1   1   0   1
|       |       |   |       |
|       |       |   |       1 ×2⁰=   1
|       |       |   |_________×2²=   4
|       |       |_____________×2³=   8
|       |_____________________×2⁵=  32
|_____________________________×2⁷= 128
                                   ---
                                   173
```

En el BASIC también podemos indicar directamente los números binarios, poniendo delante del número el prefijo &B.

Ejemplo:

```
10 A%=&B10101101
20 PRINT A%
```

Resultado:

```
173
```

Números octales y hexadecimales

Depués de todo lo que llevamos diciendo, podemos indicar simplemente lo que son números octales y números hexadecimales.

Octales son los números cuyas cifras, de acuerdo con nuestra explicación anterior, se asocian con las potencias de 8. En los hexadecimales, la posición de cada cifra guarda relación con una potencia de 16.

Es curioso constatar que en los números decimales tenemos exactamente 10 cifras (0/9) para representar estos números. En los números binarios sólo hay 2 cifras (0 y 1) y en los octales tenemos 8 (0/7), como es natural.

Hasta aquí no hay problema. En los hexadecimales se habla de 16 cifras, claro; pero ahora se nos ocurre que no conocemos más que las cifras 0 a 9. ¿Cómo indicamos, entonces, el resto de las cifras en los números hexadecimales? Pues, para ello, utilizamos las 'letras' A a F y todo solucionado.

La tabla siguiente ofrece los números decimales de 0 a 15. Estos, para que sirva de ilustración, se representan también en forma binaria, octal y hexadecimal.

decimal	*binaria*	*octal*	*hexadecimal*
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Por el momento, parece que todo esto no sirviera para nada. Sin embargo, más adelante (en el cápitulo sobre los *sprites*, entre otros) veremos la utilidad que tiene.

Los números octales y hexadecimales pueden incluirse directamente en los programas. Para esto, colocaremos delante de un octal, &O y delante de un hexadecimal, &H.
Ejemplos:

```
PRINT &O17 da 15
PRINT &HF  da 15
PRINT &HFF da 255
```

6

PRINT, TAB, LOCATE, PRINT USING Y REM

En capítulos anteriores hemos visto ya, que detrás de la instrucción PRINT puede colocarse más de un dato a exponer.
Veamos un ejemplo:

```
10 A=12
20 PRINT "A=";A
30 PRINT "A=",A
```

Resultado:

```
A=12
A=            12
```

La diferencia entre el resultado de la línea 20 y el de la 30 se encuentra en el signo de separación. En la línea 20, hemos utilizado el punto-y-coma como signo de separación y en el renglón 30, la coma.

Con el punto y coma, el resultado que hay que exponer a continuación, sigue inmediatamente al anterior. Si utilizamos la coma, en cambio, el ordenador efectúa automáticamente un reparto en columnas.

En este reparto en columnas se tiene siempre en cuenta un espacio intermedio de 14 posiciones.

Estos signos de separación también pueden colocarse al final de la instrucción PRINT, como nuestra el ejemplo siguiente:

```
10 PRINT "ABC"; "DEFG";
20 PRINT "H"; "IJ"; "KLM"
```

Resultado:

```
ABCDEFGHIJKLM
```

Si, al exponer un texto, quiere omitirse un línea, se utilizará una instucción PRINT sin indicar ningún dato a exponer.

Ejemplo:

```
10 PRINT "A"
20 PRINT
30 PRINT "B"
```

Resultado:

```
A

B
```

Está claro que, entre A y B, hay un línea en blanco.

TAB Por medio de la función TAB podemos indicar, también, en qué columna tiene que colocarse una magnitud cuyo valor exponemos.
El siguiente ejemplo aclarará el uso de TAB.

```
10 PRINT TAB(1); "MSX"
20 PRINT TAB(3); "MSX"
30 PRINT TAB(5); "MSX"
```

Resultado:

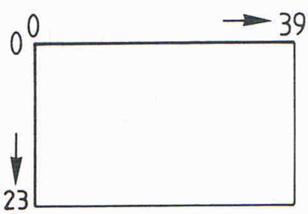
```
MSX
  MSX
    MSX
```

Vemos aquí cómo después de TAB se indica, entre paréntesis, la posición en la columna a patir de la cual debe ser colocado lo que haya que exponer (en este caso un texto).

LOCATE La función TAB sólo regula la posición de exposición dentro de la línea. LOCATE por el contrario indica tanto la posición horizontal como la vertical. Es una instrucción muy fuerte, por medio de la cual podemos conseguir, de forma fácil, una hermosa ordenación para el texto expuesto en la pantalla.
La instrucción LOCATE funciona así:

LOCATE posición horizontal, posición vertical

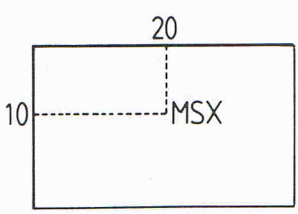
Esta es la división correspondiente de la pantalla:



Ejemplo de programa:

```
10 CLS
20 LOCATE 20,10
30 PRINT "MSX"
```

da como resultado:



El programa comienza con una instrucción no explicada todavía: la instrucción CLS (Clear-Screen). El resultado de ésta es que la pantalla se borra y el cursor se coloca en el ángulo superior izquierdo. La instrucción LOCATE, puesta a continuación, sitúa el cursor en la posición (20, 10).

Esto quiere decir que el texto MSX se expondrá a partir de esa posición.

WIDTH Notemos ahora que con esta instrucción se puede determinar el número de posiciones horizontales a utilizar. Para una descripción detallada de las instrucciones SCREEN y WIDTH, véase el Resumen de las instrucciones MSX-BASIC

PRINT USING En muchas aplicaciones nos interesa que los resultados aparezcan en una forma determinada. Como ejemplo ilustrativo podemos pensar en cuentas financieras, en las que los resultados tienen que ser presentados siempre con dos cifras detrás de la coma.

La manera de exponer así un número se indica con la instrucción llamada PRINT USING.

Esta se construye siempre de la siguiente forma:

PRINT USING 'información sobre como exponer'; número

Daremos un ejemplo:

```
PRINT USING "##.##";32.7
```

En la pantalla aparecerá:

```
32.70
```

La expresión puesta entre comillas, o sea, ##.## indica cómo debe ser presentado el número. En este caso, con dos cifras delante del punto decimal y dos cifras detrás.

```
                 ##.##
            ↗      ↑      ↖
dos cifras    el punto    dos cifras
   delante                detrás
 del punto                del punto
```

Las posibilidades disponibles con repecto a PRINT USING son excepcionalmente numerosas.

En el Resumen de las instrucciones MSX-BASIC se presenta un sumario completo.

REM Esta es la última instrucción que comentaremos en este capítulo. El término REM proviene de 'remark', que quiere decir 'observación'. Con ella podemos añadir comentarios a un programa. Estos comentarios debemos ponerlos siempre detrás del término REM.
Un ejemplo:

```
10 REM ESTE NO ES MAS QUE UN EJEMPLO
20 PRINT "FIN"
```

Resultado, después de dar el comando RUN:

```
FIN
```

Vemos que el comentario situado detrás de REM, no se expone al dar el comando RUN. Sólo podemos verlo aparecer, si damos el comando LIST.

7 INSTRUCCIONES DE CONTROL: GOTO, IF...THEN, FOR...NEXT Y ON...GOTO

GOTO

La instrucción GOTO tiene una forma muy sencilla:

```
GOTO número de línea
```

Cuando el ordenador se encuentra con esta instrucción, da un salto hasta la línea, cuyo número está mencionado explícitamente y luego, el programa prosigue a partir de ella. En vista de que el salto *siempre* se ejecuta hablaremos de esta instrucción como la de salto *incondicional.*



Un ejemplo sencillo:

```
10 PRINT "ESTO SERA EXPUESTO"
20 GOTO 40
30 PRINT "ESTO NO"
40 PRINT "ESTO TAMBIEN SERA EXPUESTO"
```

Resultado:

```
ESTO SERA EXPUESTO
ESTO TAMBIEN SERA EXPUESTO
```

La línea 20 hace dar un salto hasta la linea 40, de modo que la ejecución de la línea 30 es omitida siempre.

IF...THEN

La forma más sencilla de la instrucción IF...THEN es:

```
IF condición THEN instrucción
```

Vemos que, entre el término IF y el término THEN, hay que mencionar una condición; una condición tal que SI (IF) se cumple, ENTONCES (THEN) se ejecutará la instrucción indicada. Lo mejor es que aclaremos esto con un ejemplo muy sencillo:




```
10 INPUT "TECLEA UN NUMERO";K
20 IF K=3 THEN PRINT "EL NUMERO ERA TRES"
30 PRINT "FIN DEL PROGRAMA"
```

El programa comienza con una instrucción INPUT, que indica que hay que ingresar un número por el teclado. Imagine que inscribimos el valor 3: o sea, a K se le impone el valor 3. En la linea siguiente se encuentra una instrucción IF...THEN.
La condición es:
K=3
o, en palabras:
¿Es K igual a 3?

Observemos que hemos formulado la condición en forma de pregunta. Pues bien, una condición así sólo puede ser correcta o incorrecta. Es decir, en relación con el ejemplo: K es ciertamente igual a 3, o K no es igual a 3.

Entonces nos referimos a que la condición es o no es *cierta*. O, para decirlo de un modo más elegante, la 'expresión lógica' es o no es cierta. A K se le había impuesto el valor 3 y la conclusión es, por consiguiente, que se ha cumplido la condición.

En este caso, el ordenador ejecutará la instrucción encabezada por THEN con el resultado de exponer en pantalla la frase 'EL NUMERO ERA TRES'. Si K no fuera igual a 3; en otras palabras en todos los demás casos en que no se cumpla la condición, la instrucción sería, entonces, ignorada a partir de THEN. Cuando el ordenador haya cumplido la instrucción IF...THEN, continuará ahora con la instrucción siguiente.

En el ejemplo, que ponemos ahora a continuación, mostramos un programa en el que, después de THEN, aparece una instrucción GOTO:

```
10 PRINT "¿CUANTO ES 2+5?"
20 INPUT K
30 IF K=7 THEN GOTO 80
40 PRINT "NO ES CORRECTO"
50 PRINT "EL RESULTADO DE 2+5"
60 PRINT "ES 7 NATURALMENTE"
70 GOTO 90
80 PRINT "ESA ES LA RESPUESTA CORRECTA"
90 END
```

Vemos aquí que después de THEN ha sido incluída la instrucción GOTO 80. Así pues, si se le da a K el valor de 7, se efectuará el salto hasta la línea 80.

Si para K se teclea un valor distinto de 7, esta instrucción GOTO será omitida y entonces les toca el turno sucesivamente a las lineas 40, 50, 60 y 70. Observe que la línea 70 indica, de nuevo, una instrucción GOTO. Esta es necesaria porque si no, después del texto de los renglones 40, 50 y 60, aparecería el texto mandado exponer en la línea 80..., y esto parecería, cuando menos, un poco tonto.

La última instrucción de este programa es la llamada instrucción END, que obliga a que el programa finalice.

Esta instrucción puede omitirse, si se quiere. De hecho, ya hicimos esto en los programas anteriores.

A propósito de este programa haremos las siguientes observaciones:

- En lugar de IF K=7 THEN GOTO 80
  podía haberse escrito
  IF K=7 THEN 80
  o mejor omitiendo THEN
  IF K=7 GOTO 80

- Detrás de THEN, podíamos haber puesto más instrucciones, separadas por el signo: , a condición de que toda la frase IF...THEN cupiera en una sola linea de BASIC (255 símbolos como máximo).

- En la instrucción IF...THEN, K se compara con 7 con el signo =. Un signo así, en una condición, decimos que es un signo de relación. Aparte del signo = podemos utilizar también los signos siguientes:

signo	*significado*
<	menor que
>	mayor que
<=	menor que o igual a
=<	idem
>=	mayor que o igual a
=>	idem
<>	distinto de
><	ídem

- También está permitido combinar expresiones lógicas. Las posibilidades existentes se comentan en el Apéndice F.

IF...THEN...ELSE

En el BASIC está también permitida la instrucción IF...THEN, ampliada con el término ELSE. Ilustraremos esta ampliación con un ejemplo sencillo:

```
10 INPUT K
20 IF K=7 THEN PRINT "K ES 7" ELSE
               PRINT "K ES DISTINTO DE 7"
30 END
```

La instrucción detrás del THEN será ejecutada sólo si se cumple la condición previa (K=7); *en* todos los *demás* casos, se ejecutará la instrucción después de ELSE. Aquí también ocurre, como en THEN, que podemos incluir más instrucciones, separadas por el signo dos-puntos.

FOR...NEXT

La instrucción FOR...NEXT tenemos que considerarla como un instrumento de programación muy útil, con el cual podemos indicar que una serie determinada de instrucciones deben ser repetidas cierto número de veces. El ejemplo siguiente aclarará esto:

```
10 FOR A=1 TO 5
20 PRINT A; "ESTA ES UNA DEMOSTRACION"
30 NEXT A
```

Resultado:

```
1 ESTA ES UNA DEMOSTRACION
2 ESTA ES UNA DEMOSTRACION
3 ESTA ES UNA DEMOSTRACION
4 ESTA ES UNA DEMOSTRACION
5 ESTA ES UNA DEMOSTRACION
```

Las instrucciones que deben ejecutarse repetidas veces se deben intercalar entre la línea que empieza por FOR y la línea que termina por NEXT, o sea, esquemáticamente:

```
.. FOR     nombre de variable=...
.. ...  }  estas instrucciones se ejecutan
.. ...  }  repetidamente las veces indicadas
.. ...  }
.. NEXT    nombre de variable
```

En nuestro caso, sólo se trata de repetir una instrucción, es decir la de la línea 20. Puede determinarse cuántas veces serán ejecutadas las instrucciones, por lo que está descrito en la línea que empieza por FOR.

En

```
FOR A=1 TO 5
```

Será repetida exactamente 5 veces, aquellos en que A toma, sucesivamente, los valores 1, 2, 3, 4 y 5. La variable que se indica después de FOR se llama, muy oportunamente, 'variable contador'.

En este caso, A se va aumentando en uno cada vez. También podemos dar *pasos* más grandes añadiendo a la línea de FOR el término STEP:

```
10 FOR A=1 TO 6 STEP 2
20 PRINT A; "ESTA ES UNA DEMOSTRACION"
30 NEXT A
40 PRINT A
```

Resultado:

```
1 ESTA ES UNA DEMOSTRACION
3 ESTA ES UNA DEMOSTRACION
5 ESTA ES UNA DEMOSTRACION
7
```

Vemos cómo la variable de contador A obtiene aquí, consecutivamente, el valor de 1,3 y 5 y cómo se sale de 'el bucle' por el valor pertinente.

Si A es igual a 7, se sobrepasará el límite estipulado en la línea 10, por lo que el ordenador continúa con la línea 40. Hacemos exponer de nuevo el valor de A y, así, se imprimirá el valor de 7.

La moraleja de este programa es clara...¡Hay que tener cuida-

do al utilizar una instrucción FOR...NEXT al seguir usando la 'variable contador' en el programa; ésta no tiene, necesariamente, el valor de finalización que está indicado en la instrucción FOR.

Algunos ejemplos más:

- `FOR A=10 TO 5 STEP-1`
  ahora es ejecutada la serie de instrucciones *para* A=10, 9, 8, 7, 6 y 5

- `FOR A=-15 TO 15 STEP 3`
  ahora es ejecutada la serie de instrucciones *para* A=-15, -12, -9, -6, -3, 0, 3, 6, 9, 12 y 15

- `FOR A=1.4 TO 1.7 STEP .05`
  ahora es ejecutada la serie de instrucciones *para* A= 1.4, 1.45, 1.50, 1.55, 1.60, 1.65 y 1.70

- `FOR A=B TO C STEP K/L`
  Vemos aquí cómo son indicados por variables, tanto el valor de comienzo como el valor de finalización de la 'cuenta' así como el tamaño del paso. El tamaño del paso se indica incluso con una expresión aritmética (división de dos variables). Pues sí, está permitido indicar los valores de comienzo, finalización y de paso por medio de esta clase de expresiones.

- Las instrucciones FOR...NEXT pueden contener, a su vez, otras instrucciones FOR...NEXT. La condición es que una instrucción englobe completamente a la otra. Así, la construcción sintáctica:

```
FOR K=1 TO 10
...
FOR J=1 TO 5   }
...            } el bucle interior está rodeado
...            } completamente por el bucle exterior
NEXT J         }
NEXT K
```

está permitida, y la construcción sintáctica:

```
FOR K=1 TO 10  }
...            }
FOR J=1 TO 5   } "bucle K"
...            }              }
...            }              }
NEXT K         }              } "bucle J"
...                           }
NEXT J                        }
```

está prohibida. Pues aquí el bucle interior no está completamente englobado por el bucle exterior.

ON...GOTO La instrucción ON...GOTO es la última que trataremos en este capítulo. Esta instrucción hace pensar en un *conmutador de selección* y un salto SEGUN......

En base al valor que se encuentra entre ON y GOTO se ejecuta un salto determinado.
Un ejemplo hará esto más claro:

```
10 INPUT K
20 ON K GOTO 30, 50
30 PRINT "K ES 1"
40 GOTO 70
50 PRINT "K ES 2"
60 GOTO 70
70 END
```

Si K es igual a 1, se dará un salto hasta el primer número de línea mencionado. Si K es igual a 2, sigue un salto hasta el segundo número de línea mencionado en la instrucción.

En este ejemplo sólo se han mencionado dos números de línea después de GOTO (línea 20). De hecho, pueden ser muchos más. Fijémonos asimismo en que las diferentes partes del programa, a las que se salta, se cierran a su vez con una instrucción GOTO. El uso de las instrucciones GOTO es imprescindible en este caso.

8 TABLA ('ARRAY')

Introducción

En muchos programas hay necesidad de disponer de gran número de variables. Podríamos pensar, por ejemplo, en un programa para la administración de existencias en un almacén. Si para cada artículo almacenado, tenemos que reflejar en el programa una variable diferente, el programa será, desde luego, muy grande.

En lo que vamos a tratar a continuación veremos cómo se solucionan en BASIC estos problemas, ofreciendo la posibilidad de trabajar con gran número de variables, homogéneas por medio de un sólo nombre.

Instrucción DIM

La instrucción DIM es aquella por la cual se reserva espacio para un gran colectivo de variables. Esta tiene siempre la forma siguiente:

DIM nombre (cantidad)

por ejemplo:

```
DIM A (100)
```

Con esta instrucción reconoce BASIC una serie de 101 variables. El *nombre* de cada una de estas variables se compone del nombre colectivo que aparece en la instrucción DIM seguido de un número entre paréntesis, que se llama subíndice.

Así:

```
A(0) A(1) A(2) A(3) ... A(99) A(100)
```

constituyen exactamente 101 variables que pertenecen a la instrucción DIM A(100). En un caso así hablamos, a veces, de la tabla A, que en este ejemplo se compone de los elementos A(0) hasta A(100).

Todas esas variables podemos utilizarlas de la misma manera que las variables elementales tratados hasta ahora.

Un ejemplo:

```
10 DIM A(100)
20 A(3)=6
30 A(27)=5
40 A(98)=A(3)+A(27)
50 PRINT A(98)
```

Resultado:

```
11
```

Este programa, que en sí parece un poco raro, muestra que hemos utilizado los elementos de matriz A(3), A(27) y A(98) de la tabla colectiva, como si se tratara de variables corrientes llamadas, por ejemplo: A, B y C.

Las posibilidades de las variables colectivas están determinadas, sobre todo, por el hecho de que podemos controlar el número que va entre paréntesis -el subíndice- para señalar un elemento cualquiera perteneciente al colectivo.

Ejemplos:

A(93)	el subíndice se indica directamente con un número
A(K)	el subíndice se indica indirectamente con una variable
A(K+3)	el subíndice se indica indirectamente con una expresión

El ejemplo siguiente lo ilustra de una manera sencilla:

```
10 DIM B(20)
20 FOR K=1 TO 20
30 B(K)=K
40 NEXT K
50 FOR K=20 TO 1 STEP -1
60 PRINT B(K);
70 NEXT K
```

Resultado: 20 19 18 17 16 15 14 13 12
11 10 9 8 7 6 5 4 3 2 1

En la línea 10 se 'dimensiona' la tabla B. Las lineas 20/40 indican que a B(1) hay que asignarle el valor 1, a B(2) el valor 2, etc. La prueba de que se han asignado realmente esos valores la conseguimos con las lineas 50/70 que tienen como consecuencia la exposición de los valores de las variables B(20), B(19), B(18), etc. Observemos asimismo lo cómodo que resulta el poder manejar la instrucción FOR...NEXT en combinación con las tablas.
A propósito de las tablas haremos las siguientes observaciones:

- Si en un programa se utiliza una variable subindicada A(6) por ejemplo, sin que previamente se haya hecho mención de la instrucción DIM, el ordenador la reconoció como elemento de una tabla con un límite superior de 10. En el ejemplo, pues, el ordenador parte del supuesto de que la variable A(6) pertenece a la tabla A que tiene 10 como límite superior (DIM A(10)).
- También podemos utilizar en un programa las llamadas tablas pluridimensionales, es decir, entre los paréntesis se meterán ahora más subindices. Así, la instrucción DIM A(3,3) preparará las variables:

```
A(0,0)        A(0,1)        A(0,2)        A(0,3)
A(1,0)        A(1,1)        A(1,2)        A(1,3)
A(2,0)        A(2,1)        A(2,2)        A(2,3)
A(3,0)        A(3,1)        A(3,2)        A(3,3)
```

- En una linea pueden darse las dimensiones de varias tablas Por ejemplo:

```
10 DIM A(100), P(300), Z(50), PI(30), B%(5)
```

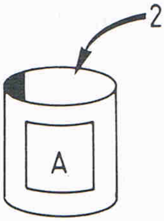
- Ulteriormente, podemos liberar si es necesario, el espacio reservado, por medio de una instrucción ERASE.
  Ejemplo: `ERASE A,P`

9 LITERAL...JUGAR CON LETRAS

Introducción

Hasta ahora hemos visto que a una variable le podemos asignar un valor numérico, al explicar cómo se hacía esto, hicimos uso de una metáfora con una gaveta.

Así se hacía evidente la instrucción A=2 por medio de la siguiente figura:



A las variables también les podemos asignar como valores *series de letras*, formando una retahila o 'letrero'.. A estas valores los llamanos literales y es por eso que a estas variables las llamamos variables literales, (por contraposición los otros: los numerales).

Desde luego, en la manera de escribirlas, tendremos que dejar claro que se trata de variables literales. El convencionalismo es sencillo: Si inmediatamente detrás del nombre de una variable ponemos el signo del dólar $, es que se trata de una variable literal.

Así, tenemos A$ PRIMER$ TEXTO$, como ejemplos de nombres que se refieren a variables literales.

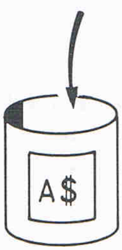
El texto, que podemos asignar como valor de una variable literal, debe ir siempre *entre comillas.*

El programa siguiente muestra un ejemplo:

```
10 LET A$="ESTE ES UN LETRERO"
20 PRINT A$
```

La línea 10 podemos aclararla por medio de un dibujo.

ESTA ES UNA CADENA
texto que se guarda en A$



Observemos que, en la linea 10, el texto se ha colocado entre comillas. Si ahora miramos el resultado del programa, nos llamará la atención que las comillas no se exponen en pantalla. Como resultado aparece:

```
ESTE ES UN LETRERO
```

Por lo tanto podemos concluir que las comillas no forman parte de los datos literales. Solo sirve para delimitarlo.

A continuación siguen algunos ejemplos con datos literales que podemos asignar como valores a variables de la cadena literal.

`"HOLA JUAN"` éste es un literal que consta de nueve simbolos.
¡Atención! El espacio que hay entre HOLA y JUAN también se cuenta.

`"164"` éste es un literal que conta de tres símbolos. El hecho de que los símbolos sean cifras es engañoso porque el ordenador ve sólo 'cadenas de caracteres' sin fijarse si son cifras o no. Veremos cómo, en estos casos especiales, podemos cambiar un 'pseudo número' de éstos en un número de verdad con los que se pueden hacer operaciones matemáticas.

" " — éste es un literal que no consta de ningún caracter pues las comillas están colocadas unas inmediatamente al lado de las otras. Luego veremos ejemplos de las ventajas que puede ofrecer este literal vacío. Este literal se designa, a veces, con el término literal nulo.

Concantenar... o empalmar literales

Por medio del signo de sumar + se puede indicar que dos literales deben ser colocados uno tras otro. Los expertos llaman a esto *concatenar*.

Un ejemplo aclarará esto:

```
10 A$="MSX-"
20 B$="BASIC"
30 C$=A$+B$
40 PRINT C$
```

Resultado:

```
MSX-BASIC
```

Vemos que en las líneas 10 y 20 los literales 'MSX-' y 'BASIC' han sido asignados a A$ y a B$. En la línea siguiente se 'suman' los literales y se asignan a C$ una vez concatenados.

Funciones literales

Hasta ahora aún no hemos podido hacer mucho con los literales, a lo sumo, unirlos entre sí para conseguir uno más largo.

Pero, afortunadamente el BASIC incorpora una gran cantidad de funciones con las que se pueden ejecutar toda clase de acciones relacionadas con varibles y valores literales.

Podemos dividir estas funciones literales en dos grupos:

Grupo 1 Las funciones que tienen como resultado *otro literal*. Los *nombres* clave de estas funciones terminan siempre por $.

Grupo 2 Las funciones que tienen un *número* como resultado y *un literal* como argumento. La mayor parte de estas funciones opera de una manera determinada de antemano. Esta operación tiene que ver con la numeración interna de los *caracteres* en el ordenador. Los nombres para las funciones de este grupo no acaban con el signo del dólar.

Ambos grupos de funciones son, naturalmente, muy sencillos. En el resumen de las instrucciones MSX-BASIC presentamos una descripción detallada, y se presenta un ejemplo de cada una de ellas. Nos es suficiente dar aquí una corta enumeración de las funciones más importantes.

- LEN — determina el número de caracteres en una cadena
- LEFT$ — nos presenta una parte de una cadena, el así llamado subliteral (parte izquierda).
- RIGHT$ — nos presenta un subliteral (parte derecha).
- MID$ — nos presenta un subliteral (en general).
- ASC — nos presenta el valor ASCII del primer carácter.
- CHR$ — nos presenta el carácter, de acuerdo al valor ASCII introducido.

VAL convierte un literal numérico en el número correspondiente.
STR$ convierte un número en el literal númerico correspondiente.
SPACE$ sirve para la exposición de un número determinado de espacios.
INSTR se utiliza para buscar un subliteral en una cadena.

INKEY$ y un juego

INKEY$ no es, en realidad, una función sino una variable. Cuando el ordenador se encuentra con el término INKEY$ durante la ejecución de un programa, inspecionará el teclado; si en ese momento está pulsada una tecla, ese caracter quedará asignado a INKEY$. Si en ese momento no pulsamos ninguna tecla, se asignará a INKEY$ el literal vacío "".

El programa siguiente servirá para ilustrar cómo puede usarse INKEY$ para hacer un juego de agilidad:

```
10 PRINT "PULSE UNA TECLA"
20 PRINT "CUANDO LA PANTALLA"
30 PRINT "MUESTRE EL VALOR 25"
40 FOR K=1 TO 50
50 PRINT K
60 A$=INKEY$
70 IF A$<>"" THEN GOTO 90
80 NEXT K
90 PRINT "FIN"
```

Algunas observaciones

Hasta ahora hemos explicado las funciones más importantes en relación con los literales. En el resumen de las instrucciones MSX-BASIC se encontrará un cuadro con todas las instrucciones posibles.
Finalmente observaremos que:

- Del mismo modo que ocurría con los númerales, tenemos la posibilidad de introducir tablas de literales, por ejemplo:

```
DIM A$(100)
```

- El ordenador, al ponerse en marcha, tiene una cantidad determinada de memoria reservada para el almacenamiento de datos literales. De modo estándar pueden almacenarse 200 caracteres. Este espacio de memoria lo podemos ampliar con la llamada instrucción CLEAR, por ejemplo:

```
CLEAR 500
```

Con la instrucción CLEAR reservamos el espacio necesario para el almacenamiento de 500 caracteres. (y 'clear' significa 'aclarar' dejar en limpío)

- El literal que nos esta permitido asignar a una variable de literal tiene una longitud máxima de 255 caracteres.

10 SUBRUTINAS: GOSUB ... RETURN Y DEF FN

GOSUB...RETURN

Una subrutina es una parte de un programa a la cual se puede llamar desde cualquier parte del mismo. El salto a la subrutina es siempre por medio de la instrucción:

```
GOSUB número de línea
```

Cada vez que el ordenador encuentre esta instrucción, continuará el programa desde el número de línea indicado. Después de reconocer el termino

```
RETURN
```

regresará el ordenador al punto de partida del programa. El esquema seguiente ilustrará esta situación.

```
Programa principal            subrutina
 10  ...
 20  ...                      300  ...
 ..  ...                       ..  ...
 ..  ...                       ..  ...
100 GOSUB 300                  ..  ...
110  ...                      ...  RETURN
```

Vemos cómo se llama la subrutina desde la línea 100. Esta empieza en la línea 300.

En el momento en que se encuentre la instrucción RETURN, regresará el ordenador a la línea 110, es decir, el ordenador continuará el programa nuevamente desde el punto de partida; observemos aquí que desde una subrutina siempre es posible saltar hacia otras subrutinas.

El siguiente programa nos ilustrará ésto. Con éste programa podemos jugar al conocido juego de 'las 13 cerillas'. Podemos coger por turno, 1, 2, o 3 cerillas de una pila. Pierde quien coja la última cerilla.

```
 10 PRINT "USTED EMPIEZA"
 20 L=13
 30 GOSUB 80
 40 GOSUB 80
 50 GOSUB 80
 60 PRINT "JA, JA, JA, YO HE GANADO"
 70 END
 80 REM COMIENZO DE LA SUBRUTINA
 90 INPUT K: IF K<0 OR K>3 THEN GOTO 90 ELSE A=4-K
100 PRINT "YO COJO ... ";A
110 L=L-K-A
120 PRINT "QUEDAN TODAVIA ....";L
130 PRINT "AHORA LE TOCA NUEVAMENTE A USTED"
140 RETURN
```

Observemos como por medio de END se ha separado la subrutina del programa principal. Así nos evitamos entrar nuevamente en la subrutina sin hacer uso de la instrucción GOSUB. Aclaremos ésto: sólo está permitido llegar a una subrutina utilizando la instrucción GOSUB.

DEF FN Con ayuda de la instrucción DEF FN podemos definir nosotros mismos una función.

La instrucción DEF FN tiene siempre esta forma:

DEF FN nombre (serie de variables separadas por comas)= expresión en que aparecen esas variables.

Ejemplo:

```
DEF FNA(X,Y)=X^2+Y^2
```

El nombre de la función consta de una sola letra; en este caso, la letra A. En este ejemplo la función A está definida por la formula:

X^2+Y^2

Esta función se aplica mencionando en cualquier expresión la FNA es decir, con el término FN seguido del *nombre* de la función.

El programa siguiente lo ilustrará:

```
10 DEF FNA(X,Y)=X^2+Y^2
20 A=3
30 B=4
40 C=FNA(A,B)
50 PRINT C
```

Resultado:

```
25
```

EXPANSION DEL MSX-BASIC

1 SONIDO: BEEP, PLAY Y SOUND

BEEP Quiere decir algo así como 'pitido'.
El programa siguiente illustrará esta instrucción:

```
10 PRINT "COMIENZO"
20 FOR K=1 TO 1000
30 PRINT K
40 NEXT K
50 BEEP
60 PRINT "FINAL"
```

Vemos cómo la instrucción BEEP ha sido colocada al final del programa. Cuando el ordenador acabe sus calculos se oirá una señal acústica.

PLAY La instrucción PLAY está dirigida pura y simplemente a la generación de melodías.

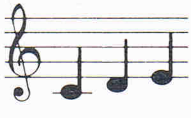
Es una instrucción muy potente en la que podemos distinguir, entre otros, los siguientes matices:

- el movimiento en el que hay que tocar
- la octava a la que pertenece una nota
- la duración de una nota
- la nota (tono) misma
- compases de silencio
- el volumen en que se toca una nota
- ciertos efectos de sonido

PLAY puede usarse como instrucción y también como comando.
Escriba:

```
PLAY "CDE"
```

Después de pulsar RETURN oiremos:



O sea, las notas do, re, mi que pertenecen a la octava de la clave de sol; indicaremos esta octava, desde ahora, como 'octava 4'.

El siguiente programita se sirve de la instrucción PLAY para convertir al ordenador MSX en un sencillo órgano.

```
10 A$=INKEY$
20 PLAY A$
30 GOTO 10
```

En la linea 10 se le asigna a A$ la tecla pulsada que, después, será usada en la instrucción PLAY, etc.

Nuestro pequeño órgano no es todavía perfecto porque, entre otras cosas, no podemos regular el movimiento. Pero, al menos tenemos la posibilidad de regular otras muchas cosas.

Mas canales de sonido

Si introducimos tres literales separados por comas, después de PLAY, el ordenador MSX utilizará cada uno de los literales para un canal de sonido independiente. Por ejemplo con:

```
PLAY "CDE"
```

se hará siempre uso de un solo canal; con

```
PLAY "CDE", "EFG"
```

se utilizarán dos canales simultáneamente y con

```
PLAY "CDE", "EFG", "GAC"
```

se hará uso de los tres canales al mismo tiempo. Esto significa que las notas C, E y G (do, mi, sol) seguidas de D, F y A (re, fa, la) y finalmente E, G Y C (mi, sol, do) se oirán simultáneamente.

Movimiento

El movimiento lo indicamos con el llamado código T. Ponga el siguente comando:

```
PLAY "T200CDE"
```

Las notas do, re, mi se tocan ahora más rápidamente que antes. Así es, porque el ordenador, si no se le da otra orden, toma 'la posición' T120 y T200 indica un movimiento más rápido. El valor T puede cubrir de 32 a 256.

Volumen

El volumen se indica con el código V. Para ilustrarlo:

```
PLAY "T200V13CDE"
```

Ahora oímos la misma melodía que en la imagen anterior pero el mando de volumen (V) parece que está en una posición más elevada. También es verdad que, el ordenador, si no se le da contraorden, toma 'la posición' V8 y la V13 en un volumen más elevado. El código V puede variar entre 0 (mínimo) y 15 (máximo).

Notas de la octava Las notas se indican con las letras C, D, E, F, G, A y B. En los semitonos, por ejemplo do sostenido, nos servimos del signo # (o +). Por ejemplo::

```
PLAY "CC#" y también "CC+"
```

hace oír consecutivamente do y do sostenido. Lo que nos preguntamos ahora es '¿cómo indicaremos el do mayor? Pues bien, el do mayor pertenece a la octava siguiente, y hace un momento hemos dicho ya, que si no hay contraorden, el ordenador supone que es la octava 4 (O4).

Para producir el do mayor tenemos, pues, que indicar 05. Ejemplo:

```
PLAY "CDE05CDE"
```

hace oír dos veces la serie CDE, pero la última se toca en una octava más alta.

El código de la octava (código O) puede ser 1 como mínimo y 8 como máximo, desprendiéndose de aquí que nuestro ordenador MSX puede tocar música en 8 octavas.

Duracion de las notas Hasta ahora, todas las notas tenían la misma duración, sin embargo, al abrir un libro de música, se da uno cuenta de que las notas pueden tener distinta duración.

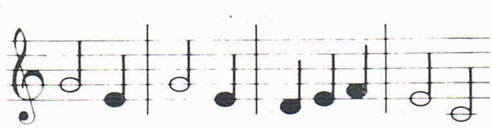
En la notación musical se indica esto rellenando o no el cuerpo de la bolita que forma la nota y, también, poniéndole banderitas al bastoncillo de la misma.

signo 

número de momentos	4	2	1	1/2	1/4	1/8	1/16
código L	1	2	4	8	16	32	64

La duración de una nota se representa con el código L. Si no se especifica más, el ordenador asume L4 y esto implica un cuarto de nota.

A continuación ilustramos el código L por medio de la siguiente tonada:



Traducida un comando PLAY sería:

```
PLAY "L2GL4EL2GL4EDEFL2EL4C"
```

Podemos acortar el número de códigos empleados si utilizamos el número de la octava immediatamente detrás del código del tono.
Así en nuestro ejemplo sería:

```
PLAY "G2EG2EDEFE2C"
```

Signos de silencio

En la notación musical también encontramos signos de silencio, éstos indican que en un período determinado, o mejor dicho, en un número determinado de espacios musicales no se pueden tocar notas. Para indicar esto, usaremos el código R.

El cuadro siguiente muestra los signos que se utilizan en la notación musical, junto con los códigos R

signo							
número de momentos	4	2	1	1/2	1/4	1/8	1/16
código R	1	2	4	8	16	32	64

El siguiente ejemplo lo ilustra:

```
10 PLAY "CDER4DECDECR2": GOTO 10
```

Después de la primera E hay un silencio de un espacio y después de la última C uno de dos espacios. Utilizando PLAY de esta manera, en un programa, puede estudiarse mejor aún el compás.

La '*hermosa melodía*' la interrumpimos con CTRL/STOP. En este párrafo sobre signos de silencio observaremos, finalmente, que detrás de una nota puede colocarse un punto. Esta nota tendrá, entonces, una vez y media más de duración, por ejemplo:

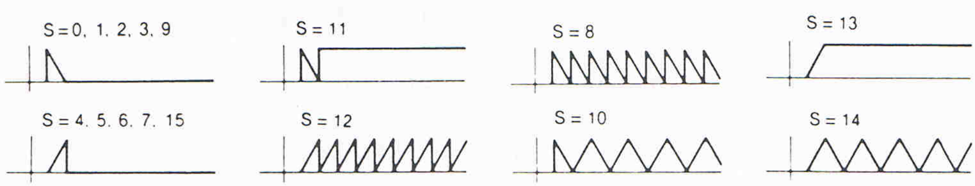
```
PLAY "CD.E"
```

De está forma podemos determinar mejor el ritmo de la música.

Los codigos S y M

Los últimos códigos que nos es permitido utiliar en la instrucción PLAY son S y M. El código S tiene que ver con un determinado timbre que podemos indicar y el M determina en qué medida ha de ser repetido el patrón de ese timbre.

El código S indica en qué proporción debe ser amplificado o disminuido el sonido. Para ello nos servimos de las siguientes imágenes:



El código M. reproduce, entonces, la longitud del ciclo, en que hay que repetir la forma que hemos elegido con S. Si no se hace ninguna otra indicación, el ordenador toma 1 como valor de S y 255 como valor de M (M puede tener entre 1 y 65535).

```
PLAY "S10M510CDE"
```

y

```
PLAY "S8M6000CDE"
```

En el primer ejemplo escuchamos un sonido fuertemente compasado, mientras que en el segundo parece como si se tocara al piano. Para darse cuenta de las diferentes posibilidades de S y M ensaye varies combinaciones para descubrirlas.

Un ejemplo desarrollado

El siguiente ejemplo hace tocar al ordenador el canon 'Frère Jacques'. En él puede verse cómo se construyen de antemano con las variables literales de que se sirve la instrucción PLAY.

```
 10 CLEAR 500
 20 A$="L4CDECR64CDEC"
 30 B$="EFG2EFG2"
 40 C$="L8GAGFL4ECL8GAGFL4EC"
 50 D$="CO3GO4C2C4O3GO4C2"
 60 W$=A$+B$+C$+D$
 70 W1$=W$
 80 W2$="R1R1"+W$
 90 W3$="R1R1"+W2$
100 PLAY W1$,W2$,W3$
```

(Nota: en el litéral de la línea 50 se utilizan letras O mayúsculas)

La instrucción CLEAR se usa para reservar suficiente espacio de almacenamiento de literales. En las líneas 20, 30, 40 y 50 se indica la melodía de los pasajes característicos. La línea 60 da la combinación de los litérales, y así, de nuevo, toda la melodía. Las líneas 80 y 90 determinan la segunda y tercera voz.

SOUND

La última instrucción que trataremos en este capítulo, es la instrucción SOUND. Esta instrucción puede entenderse bien, sólo si se comprende que nuestro chip indepediente de sonido está equipado con una cantidad determinada de registros (véase el capítulo 1). Colocándo un valor determinado en esos registros, el chip producirá toda clase de sonidos. Nuestro chip de sonido está equipado con 13 *registros* que tienen los siguientes significados:

Número de registro	*se utiliza para*
0,1	la combinación de los registros 0 y 1 determina la frecuencia de tono para el canal A de sonido.
2,3	lo mismo, pero para el canal B.
4,5	lo mismo, pero para el canal C.
6	determina la tonalidad del sonido.El valor puede variar de 0 a 31.
7	determina si un canal de sonido definido debe producir sonido o un tono.
8	determina el volumen para el canal A. (valor entre 0 y 15)
9	lo mismo que 8, pero para el canal B.
10	lo mismo que 8, pero para el canal C.
11, 12	determina la tonalidad, y en concreto, la llamada modulación de un tono.
13	determina la tonalidad (patrones de crescendo y decrescendo).

para calcular la frecuencia de un tono, nos serviremos del siguiente programa auxiliar:

```
10 INPUT "FRECUENCIA DESEADA"; F
20 A=1789772.5
30 B=INT(A/(16*F))
40 C=INT(B/256)
50 D=B-C
60 PRINT D,C
```

Los valores de D y C son los valores que hay que poner en el primer y segundo registros, para obtener el tono deseado.

Imaginemos que se quiere reproducir la nota La (frecuencia 440). Nuestro programa produce los valores:

```
254      0
```

Esto conduce a las siguientes instrucciones SOUND:

```
10 SOUND 0,254
20 SOUND 1,0
30 SOUND 8,11
```

Ahora oímos la nota *La* (probemos también con PLAY'A'), el sonido podemos interrumpirlo con CTRL/STOP.

Ruido

En la práctica utilizaremos la instrucción SOUND para efectuar toda clase de efectos especiales de sonido, como el ululal de las sirenas, sonidos de explosiones etc., los siguientes ejemplos nos muestran algunas de las posibilidades que tenemos:

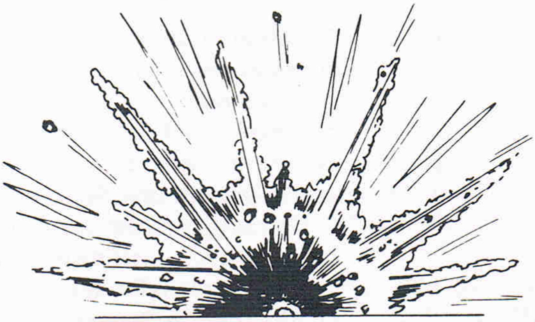


```
10 REM PITIDO DE HERVIDOR DE AGUA
20 FOR K=255 TO 0 STEP-1
30 PRINT K
40 SOUND 0,K
50 SOUND 1,0
60 SOUND 8,10
70 NEXT K
```

Si se omite la linea 30 se obtiene una variación más rápida del mismo sonido.
Con el siguiente ejemplo podemos probar algunos efectos más:

```
10 INPUT K,L,M
20 SOUND 0,250:SOUND 1,0
30 SOUND 6, K:SOUND 7,L:SOUND 13,M
40 FOR J=15 TO 0 STEP -0.05
50 SOUND 8,J
60 NEXT J
```

Con K=10, L=10 y M=10 produce esto un tono decreciente, pero con K=20, L=20 y M=20 se oye una explosión.



2 REPRESENTACIONES GRAFICAS: SCREEN, PSET, COLOR Y PRESET

Introducción

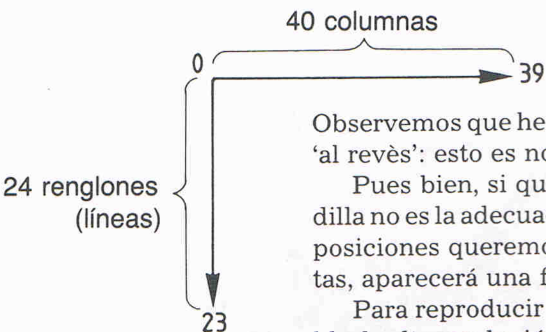
Representación gráfica no es más que una manera elegante de llamar a un dibujo. Los dibujos se construyen con puntos, lineas y arcos.

Nuestro ordenador MSX tiene una enorme cantidad de posibilidades para exhibir en la pantalla toda clase de bonitas figuras.

SCREEN

Para adquirir un poco más de visión sobre la construcción de dibujos con el ordenador MSX, debemos contar algo más acerca de las divisiones de la *pantalla* (screen en inglés).

En todas las representaciones ofrecidas hasta ahora, el ordenador adoptó una gradilla para la pantalla de 24 filas con 40 posiciones cada una



Observemos que hemos dibujado las coordenadas más o menos 'al revès': esto es normal en los ordenadores.

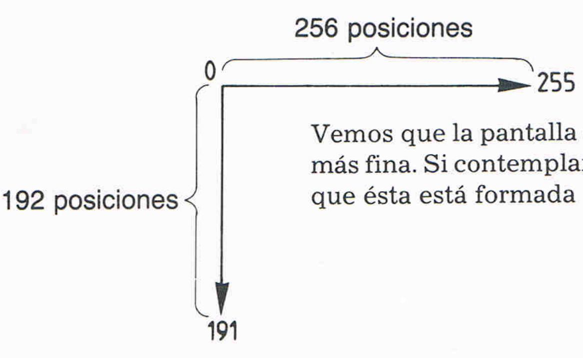
Pues bien, si queremos representar dibujos finos, esta gradilla no es la adecuada. Si, por ejemplo, en una pantalla de 24x40 posiciones queremos dibujar una línea, por medio de crucecitas, aparecerá una figura muy grande.

Para reproducir líneas de 'gran finura' (en los folletos se habla de alta resolución), tenemos que cambiar la gradilla o cuadricula de la pantalla del ordenador.

Después de la instrucción:

```
SCREEN 2
```

el ordenador adopta la gradilla interna que aparece a continuación:



Vemos que la pantalla está ahora transformada en una gradilla más fina. Si contemplamos una porción de esa gradilla veremos que ésta está formada por un entramado de cuadraditos.

Desde ahora llamaremos a cada uno de esos cuadraditos un punto de imagen (o simplemente una 'mota') e indicaremos las coordenadas de cada uno por unos valores X-e Y-.

X puede variar de 0 a 255 e Y de 0 a 191.

El punto (0,0) viene a coincidir con la esquina superior izquierda y el (191, 255) con la inferior derecha.

Después de haber instruido al ordenador, por medio de la instrucción SCREEN 2, para trabajor según esa fina gradilla podemos indicarle puntos y líneas mediante las instrucciones adecuadas para ello.

PSET Empezamos con la instrucción PSET ('pointset', es decir, *fije o ponga un punto*) por medio de la cual podemos pintar motas en la pantalla. En el caso más sencillo, esta instrucción tiene esta forma:

PSET (coordenada x, coordenada y)

Empecemos con un ejemplo:

```
10 INPUT "X=";X
20 INPUT "Y=";Y
30 SCREEN 2
40 PSET (X,Y)
50 GOTO 50
```

Las líneas 10 y 20 imponen valores a X e Y. En la línea 30 se encuentra la instrucción obligatoria para gráficos SCREEN 2. Después se dibuja un punto con la instrución PSET. (Cuando se termine el programa, se ejecutará automaticamente SCREEN 0.)

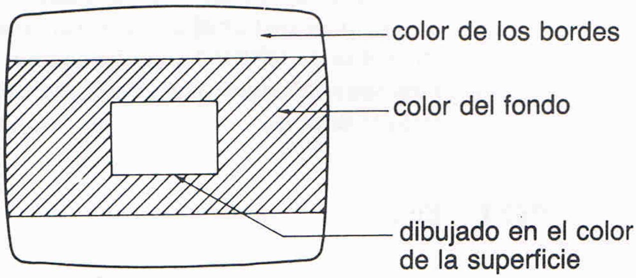
Sólo se puede interrumpir el programa pulsando STOP mientras se mantiene apretada la tecla CTRL (CTRL/STOP).

El cuadro a continuación nos presentará las diferentes posibilidades de modo-SCREEN.

modo-SCREEN	*Descripción*
0	Para exposición de texto: 40 x 24.
1	Para exposición de texto: 32 x 24.
2	Gráficos: 256 x 192 pixels ('puntos').
3	Gráficos: 64 x 48 bloques.

COLOR Antes de ver cómo podemos utilizar los colores (en inglés: COLOR), vamos a explicar algo acerca de las palabras 'color de fondo', 'color de superficie' y 'los bordes'.

La imagen siguiente expone la división de la pantalla.

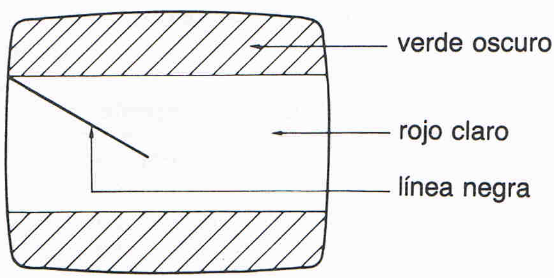


Bajo el nombre 'color de superficie' entendemos signos, símbolos, figuras, etc. que podemos exponer. Estas cosas las podemos exponer en un color determinado, y entonces hablamos de 'color de superficie'.

El color encima del cual aparecen estos signos, símbolos, figuras, etc. lo llamamos 'color de fondo'. Finalmente observamos que podemos distinguir una parte de los bordes, a la que también podemos asignar un color.
Un ejemplo:

```
10 SCREEN 2
20 COLOR 1,9,12
30 CLS
40 FOR K=1 TO 100
50 PSET (K,K)
60 NEXT
70 GOTO 70
```

Entonces aparece la figura siguiente:



Gracias a las instrucciones de las líneas 40 hasta 60 se dibujan 100 puntos que forman, juntos, una línea negra.

En este ejemplo aparece la instrucción COLOR. Esta tiene la forma siguiente:

```
COLOR    color de superficie,color de fondo,color de borde
```

Para indicar los colores tenemos los *Números identificatIVOS* (NIVOS):

NIVO/color	*NIVO/color*	*NIVO/color*	*NIVO/color*
0 transparente	4 azul oscuro	8 rojo	12 verde oscuro
1 negro	5 azul claro	9 rojo claro	13 magenta
2 verde	6 rojo oscuro	10 amarillo oscuro	14 gris
3 verde claro	7 azul cielo	11 amarillo claro	15 blanco

Así puès, en nuestro ejemplo, la línea realmente aparecerá en el color de superficie negro (1) sobre un fondo rojo claro (9), con color verde claro en los bordes (12).

Otros ejemplos más:

`COLOR 1` : coloree de negro el frente del dibujo (1), o sea dibuje todo con '*pluma*' negra

`COLOR, 2` : cambie a verde el color del fondo. Observemos que se ha colocado una *coma* delante del 2.

`COLOR1,2,11` : coloree de negro el frente o 'pluma'(1), de verde el fondo o 'papel'(2) y de amarillo claro el espacio del reborde (11).

`COLOR ,,11` : cambie el color de los espacios marginales solamente. Observemos que se han puesto dos comas delante del *Número identificatIVO de color*

Observemos finalmente que cuando queramos cambiar los colores del fondo, debemos utilizar siempre la instrucción CLS (CLear Screen) después de la instrucción COLOR.

Indicación de color detrás de una instrucción gráfica.

La instruccion PSET es una instrucción gráfica, ya que con ella podemos construir una figura. En el capítulo siguiente vamos a tratar las instrucciones LINE y CIRCLE. Estas también son instrucciones gráficas. También podemos incluir *directamente* indicaciones de colores en una de estas instrucciones gráficas como sigue:

- poniendo directamente detrás de la instrucción, el número del color. Así definimos entonces el color de superficie.

3 REPRESENTACIONES GRÁFICAS: LINE, DRAW, CIRCLE Y PAINT

Introducción

En este capítulo trataremos las demás posibilidades de reproducción de trazado de líneas en la pantalla. Las instrucciones LINE y DRAW ofrecen la posibilidad de obligar a que 'trace' líneas rectas de una manera muy simple; mientras que con CIRCLE podemos dibujar en la pantalla líneas curvas que son segmentos de circunferencias o elipses.

Todas estas instrucciones parten de la pantalla gráfica tratada en el capítulo anterior.

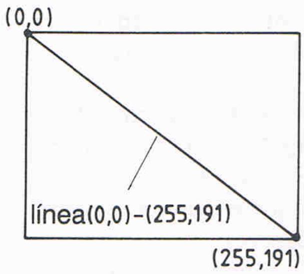
LINE

La forma más sencilla de la instrucción LINE es:

```
LINE (X1,Y1)-(X2,Y2)
```

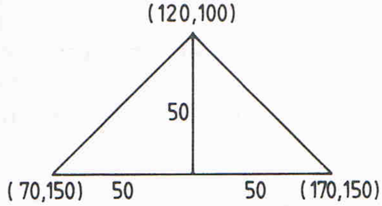
Con (X1,Y1) se indica el punto inicial del segmento de recta y con (X2,Y2) el punto final.
Ejemplo:

```
10 SCREEN 2
20 LINE(0,0)-(255,191)
30 GOTO 30
```



Como resultado vemos, ahora, una recta diagonal oblicua que recorre la pantalla.

En el programa que sigue dibujaremos un triángulo, a partir de la definición siguiente:



Programa:

```
10 SCREEN 2
20 LINE (70,150)-(170,150)
30 LINE (170,150)-(120,100)
40 LINE (120,100)-( 70,150)
50 GOTO 50
```

La posición inicial podemos eventualmente omitirla. En este caso, se toma como punto inicial el punto final que se alcanzó como consecuencia de la instrucción anterior. Veamos un ejemplo:

```
10 SCREEN 2
20 LINE ( 70,150)-(170,150)
30 LINE -(120,100)
40 LINE -( 70,150)
50 GOTO 50
```

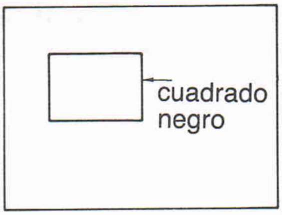
Además podemos darle un color a la línea trazada, siguiendo el mismo procedimiento que en PSET, o sea, colocando una coma y *Número identificatIVO* del color, es decir:

```
LINE (X1,Y1)-(X2,Y2), número de color
```

Con la instrucción LINE podemos trazar recuadros simples. Para ello ampliamos la instrucción con una 'B' (de Box = 'caja')

```
LINE (X1,Y1)-(X2,Y2), color, B
```

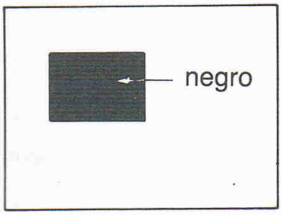
por ejemplo:




```
10 SCREEN 2
20 LINE (50,50)-(100,100),1,B
30 GOTO 30
```

Da como resultado:

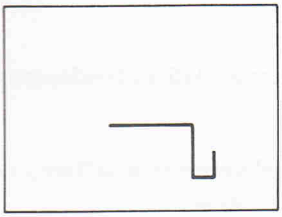
Observemos que parece que hemos definido totalmente el recuadro indicando en la instrucción LINE los dos vértices que se encuentran diagonalmente opuestos. Si en lugar de B especificamos'BF' (Fill=rellenar) obtenemos el *recuadro* de color. Por ejemplo:




```
10 SCREEN 2
20 LINE (50,50)-(100,100),1,BF
30 GOTO 30
```

DRAW Esta es una instrucción extraordinariamente potente para mandar que trace rectas horizontales, verticales y en diagonal.

Veamos un ejemplo sencillo:



```
10 SCREEN 2
20 PSET (127,95)
30 DRAW "R50D50R25U25"
40 GOTO 40
```

Como último ejemplo, trataremos un programa con el que se consigue trazar en la pantalla una maraña de líneas.

R50 trace una recta moviendo la plumilla 50 puntos hacia la derecha. (Right)

D50 trace ahora, 50 puntos hacia abajo (D viene de 'Down' que quiere decir 'hacia abajo').

R25 a continuación trace 25 puntos hacia la derecha.

U25 Y luego 25 puntos hacia arriba ('UP' significa arriba).

El cuadro completo de las posibilidades de DRAW se presenta en el Resumen de las instrucciones MSX-BASIC.

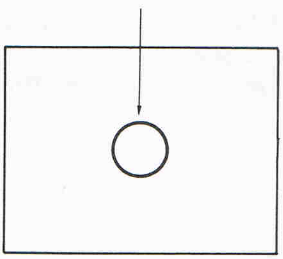
CIRCLE Esta es una instrucción para trazar circunferencias o elipses:

`CIRCLE(x,y),` radio, color, ángulo comienzo, ángulo final, curvatura o excentricidad

con:

(X,Y)	coordenadas del centro
radio	valor para el radio
color	Número identifactIVO del color
ángulo comienzo	ángulo desde el cual se dibuja el arco (expresado en radianes: 0...2π)
ángulo final	ángulo hasta el lugar en el que se dibuja el arco (expresado en radianes: 0...2π)
curvatura	número que indica si se trata de un circulo o de una elipse con más o menos excentricidad. Si se especifica aquí el valor 1.4, se obtendrá una círcunferencia en el MODO-screen 2 y 3.

círculo negro con punto central (127,95) y radio R=50



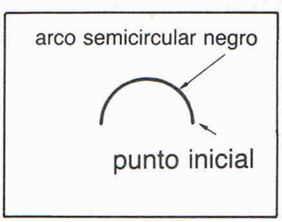
Ejemplos:

```
10 SCREEN 2
20 CIRCLE (127,95),50,1,,,1.4
30 GOTO 30
```

Resultado:

El efecto del ángulo de comienzo y de final podemos verlo con el siguiente program:

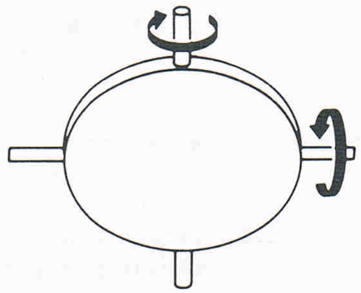
```
10 INPUT B
20 INPUT E
30 SCREEN 2
40 CIRCLE (127,95),50,1,B,E,1.4
50 GOTO 50
```



Si ahora ponemos B el valor O y E el valor 3.1415, obtendremos la siguiente semicircunferencia

El último dato de la instrucción CIRCLE determina la llamada curvatura o excentricidad, o aspecto de elipse.

Podemos imaginarnos el siguiente caso:



Si el disco gira alrededor de uno de los ejes, lo veremos cada vez bajo un ángulo distinto, es decir tendrá otra apariencia. Lo mejor es estudiar este efecto de la mano del siguiente programa:

```
10 INPUT K
20 SCREEN 2
30 CIRCLE (127,95),50,1,,,K
40 GOTO 40
```

PAINT La siguiente instrucción que trataremos en este capítulo es la instrucción PAINT. Con ella podemos colorear ciertas superficies, indicando un punto cualquiera de esa superficie.
La forma general es:

`PAINT (X,Y),` color de recinto, color de borde

con:

(X, Y)	coordenadas de un punto interno al recinto
color de recinto	color con el que se ha de pintar el area interna de la figura
color de borde	color con el que ha de estar trazada la línea límite de ese recinto

Normalmente, éste último dato de la instrucción se omite porque en la pantalla gráfica el color de la línea divisoria tiene que ser igual al de la superficie que delimita. En vista de que siempre hay que indicar el borde del recinto antes de poder proceder a colorear al área interna, el número de color, de hecho, está ya fijado.

4 SPRITES

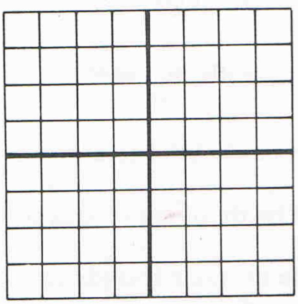
Sprite

La primera pregunta que nos hacemos en este capítulo es, naturalmente, '¿Qué es exactamente un sprite?' Pues bien, es un motivo, es una figura que podemos representar por medio de un fondo cuadriculado. A estas figuras, les damos, a continuación, un número y después, con ayuda de una instrucción especial, podemos ejecutar toda una serie de manipulaciones con ellas. La más importante es, simplemente, moverla por la pantalla.

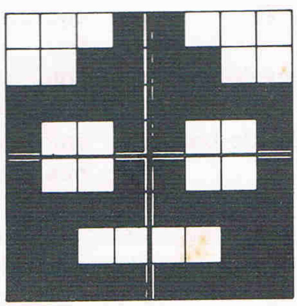
Desde luego, también podemos definir figuras por medio de instrucciones PRINT y PSET, pero para el traslado de esas figuras son necesarias muchas instrucciones, siendo el resultado que, una figurita construida así, de hecho, es intrasladable. Así que el trabajar con Sprites ofrece más posibilidades. Estos constituyen, además, una potente herramienta para que se muevan por la pantalla '*seres*' del estilo de los comecocos.

Construcción de un sprite

A continuación vamos a mostrar cómo se definen esas figuritas o 'efigies'. Para ello partimos de una matriz de 8×8.



Podemos colorear de negro ciertas casillas para obtener la figura que deseamos. Para obtener un fantasmita hacemos lo siguiente:



0001 : 1000	18
0011 : 1100	3C
1111 : 1111	FF
1001 : 1001	99
1001 : 1001	99
1111 : 1111	FF
1100 : 0011	C3
1111 : 1111	FF

A la izquierda está el fantasmita, y en la columna de la derecha vemos cómo con ayuda de unos y ceros (negro y blanco) podemos definir la misma figura.

La línea de puntos es una 'línea auxiliar': divide cada línea en dos filas de 4 unos y/o ceros. La primera línea consta, por ejemplo, de las sucesiones 0001 y 1000.

En la columna de la derecha se ve una notación con otra clase de códigos. Estos pueden derivarse directamente de la siguiente tabla. (Nota: se trata de notación hexadecimal.)

sucesión	*código*	*sucesión*	*código*	*sucesión*	*código*	*sucesión*	*código*
0000	0	0100	4	1000	8	1100	C
0001	1	0101	5	1001	9	1101	D
0010	2	0110	6	1010	A	1110	E
0011	3	0111	7	1011	B	1111	F

Con ayuda de los códigos así definidos, obtendremos un sprite siguiendo la instrucción que viene a continuación:

`SPRITE$` (número)=sucesión de instrucciones CHR$

Para nuestro ejemplo:

```
SPRITE$ (1) =CHR$(&H18)+CHR$(&H3C)+CHR$(&HFF)+
             CHR$(&H99)+CHR$(&H99)+CHR$(&HFF)+
             CHR$(&HC3)+CHR$(&HFF)
```

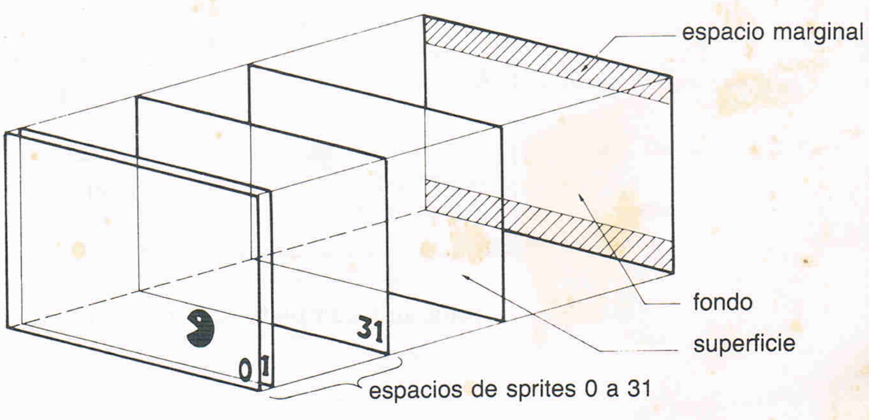
Observemos que aquí la sucesión de códigos 18, 3C, FF, etc. ha sido incorporada en las instrucciones CHR$, y cada código va precedido de los signos &H.

PUT SPRITE De esta manera que acabamos de exponer, ha sido definido por completo nuestro sprite (número 1) y, a continuación, podemos colocar la figurita en cualquier lugar de la pantalla. Para ello nos serviremos de la instrucción PUT SPRITE, que tiene la forma siguiente:

`PUT SPRITE` superficie (x, y), color, número de sprite

Aquí es:

superficie un número entre 0 y 31. Imagínese la siguiente construcción:



Es como si los sprites se movieran por superficies transparentes colocadas una detrás de la otra, como se ve arriba.

La situación de la superficie determina qué sprite resulta tapado cuando hay dos sprites que se cubren parcialmente entre sí.

(x, y) las coordenadas de la posición en que debe representase el sprite.

color el número que indica el color del sprite.

número el número que se concede al sprite por medio de la instrucción SPRITE$.

ON SPRITE GOSUB y SPRITE ON

En muchos juegos informáticos es importante saber si dos sprites se cubren parcialmente en un momento determinado o, dicho popularmente, *si entran en colisión*.

Para poder constatar esto, el MSX-BASIC ofrece la instrucción ON SPRITE GOSUB. Esta tiene la forma siguiente:

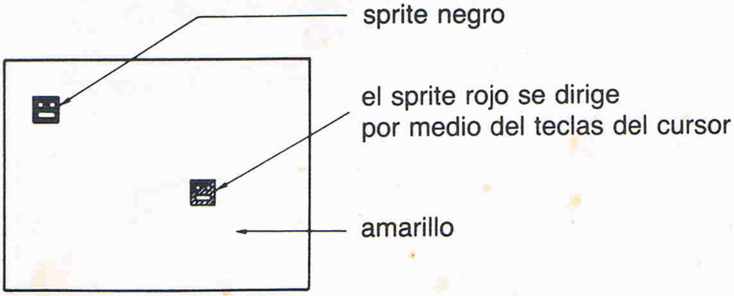
```
ON SPRITE GOSUB número de línea
```

Si el ordenador constata, ahora, que dos figuras *chocan*, salta a una *subrutina* a través del número indicado.

Pero, para ello tenemos que hacer activar la capacidad de alerta del ordenador y esto lo conseguimos por medio de la instrucción SPRITE ON.

Un ejemplo

El siguiente programa nos muestra un ejemplo. En la pantalla aparecen dos fantasmitas. Uno negro que está quieto y otro rojo que se puede mover por medio del teclas del cursor.
En imagen:



El programa completo es como sigue:

```
10 CLS
20 COLOR,11,11
30 SCREEN 2
40 X=100:Y=100
```

```
 50 FOR K=1 TO 2
 60 SPRITES(K)=CHR$(&H18)+CHR$(&H3C)+CHR$(&HFF)+
                CHR$(&H99)+CHR$(&H99)+CHR$(&HFF)+
                CHR$(&HC3)+CHR$(&HFF)
 70 NEXT K
 80 SPRITE ON
 90 PUT SPRITE 0,(40,40),1,1
100 D=STICK(0)
110 IF D=1 THEN Y=Y-1
120 IF D=2 THEN X=X+1:Y=Y-1
130 IF D=3 THEN X=X+1
140 IF D=4 THEN X=X+1:Y=Y+1
150 IF D=5 THEN Y=Y+1
160 IF D=6 THEN X=X-1:Y=Y+1
170 IF D=7 THEN X=X-1
180 IF D=8 THEN X=X-1:Y=Y-1
190 PUT SPRITE 1,(X,Y),6,2
200 ON SPRITE GOSUB 220
210 GOTO 100
220 PLAY "CCC"
230 RETURN
```

Las líneas 10, 20 y 30, no necesitan explicación. La 40 determina la posición inicial del sprite rojo. Luego, en las líneas siguientes, 50 a 70, se definen dos sprites que, por otro lado, tienen la misma forma (la que ya hemos visto antes).

Después el ordenador es colocado en posición de 'atención al choque' por medio de la instrucción SPRITE ON (línea 80). La línea 90 coloca el sprite negro en la pantalla, en la posición (40, 40). Y, a continuación, vemos la construcción de salida. Esta posición se le entrega a la instrucción PUT SPRITE de la línea 190 y ella se encarga de la colocación del sprite rojo.

La línea 200 muestra la 'pueba de choque'; si se produce una colisión, el ordenador salta a la subrutina de la línea 220.

Observaciones Cuando trabajemos con figuritas es conveniente tener en cuenta las siguientes observaciones:

- Podemos ampliar el tamaño del sprite un factor 2, sustituyendo la línea 30 por: SCREEN 2,1.
- También podemos definir sprites en una matriz de 16×16. Esta consta, pues, de cuatro bloques de 8x8 cada uno. Los bloques se numeran como sigue:

1	3
2	4

Lo más sencillo es, entonces, fijar este patrón en cuatro literales:

```
A$=CHR$()+ etc.        (definición de bloque 1)
B$=CHR$()+ etc.        (definición de bloque 2)
```

y así sucesivamente C$ y D$, y luego

```
SPRITES(1)=A$+B$+C$+D$
```

Si se trabaja con figuras de 16x16, se utilizará la instrucción SCREEN 2,2 o SCREEN 2,3, en el caso de que se desee una ampliación con factor 2.

- Con una matriz de 8x8 podemos definir como máximo 256 formas de sprite y 64 con una matriz de 16x16. No debe olvidar, sin embargo, que en cada superficie de sprite no puede representarse más de un sprite.

5 SOBRE FICHEROS

Introducción Un fichero (inglés: file) es un nombre general para una gran colección de datos. Una colección de éstas puede ser por ejemplo, un fichero de direcciones, pero también un programa en BASIC o un programa en código máquinas. Sin importarnos lo que haya en el fichero, éste tiene un nombre único con el cual podemos especificar el fichero como instrucciones o comandos. Comparemos esta situación con un armario de libros. Cada libro contiene una determiada cantidad de datos y se puede comparar con un fichero. El título del libro viene a ser el nombre del fichero.

Para qué se utilizan los ficheros?

Pues bién, si disponemos de una unidad de disco, podemos almacenar nuestros programas como si fueran ficheros.

Pero esta no es la única posibilidad que nos proporciona el MSX en relación con los ficheros.

Aquí presentamos una corta enumeración de las diferentes posibilidades que nos permite el sistema, junto con una corta aclaración que tenemos que incluir en las instrucciones.

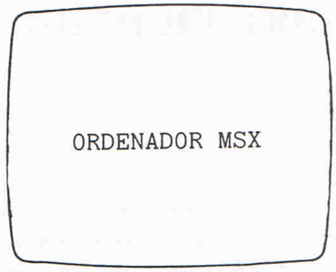
CAS La lecto-grabadora (cassette).
La lecto-grabadora puede ser utilizada como un medio barato de almacenamiento de datos y programas.

GRP La pantalla gráfica.
No está muy al alcance de la mano el utilizar la pantalla gráfica como medio de almacenamiento de ficheros. Claro está que no utilizamos la pantalla como se utiliza la lecto-grabadora. No obstante es importante saber que podemos utilizar la GRP en determinadas instrucciones, ya que así podemos exponer textos en una pantalla gráfica.

Presentamos aquí un ejemplo sencillo. Ya veremos más tarde el significado específico de las diferentes instrucciones usadas en este progama.

```
10 SCREEN 2
20 OPEN "GRP:TEST" FOR OUTPUT AS#1
30 A$="ORDENADOR MSX"
40 PRESET (50,50)
50 PRINT #1,A$
60 GOTO 60
```

En resumen, vemos textos (a saber: *ORDENADOR MSX*) en una pantalla gráfica (SCREEN 2)



CRT La pantalla para textos.
Al igual que la pantalla gráfica, podemos utilizar la 'pantalla para textos' como medio de almacenamiento para ficheros.

LPT Impresora (inglés: printer).
La impresora sirve especialmente como equipo de salida para sacar nuestros ficheros.
En el apéndice B se da una descripción detallada.

COM La interface RS232C.
Utilizando esta interface podemos acoplar nuestro ordenador a otro, por ejemplo, para enviar mensajes en ambas direcciones. Entonces se habla de 'comunicación, en lugar de 'enviar mensajes en ambas direcciones'; de ahí la denominación COM.

A hasta F Unidad de disco A hasta F.
Cada vez que, al utilizar instrucciones específicas con ficheros nombremos una de las letras A hasta F, suponemos que nos referimos a una de las unidades de disco.

La unidad de disco (inglés: disk drive) es la forma mas profesional para trabajar con ficheros. Es el único medio en el cual podemos almacenar datos de una forma predeterminada.

El nombre de un fichero El verdadero nombre de un fichero puede constar de 8 caracteres como máximo (con excepción de la lecto-grabadora: 6 como máximo). Ejemplos de nombres:

```
TEST, TEST1 y EXP1
```

Además de letras y cifras se permiten los símbolos:

```
$ & # % @ ( ) _ \ ^ { } ~ !
```

Podemos colocar un punto detrás del nombre del fichero; es la, así llamada, *extension* (a excepción de la lecto-grabadora). Así indicamos con qué clase de datos está relacionado el fichero (programas en BASIC, datos en general, etc.).
Ejemplo:

Este nombre indica que el fichero TEST se refiere a un programa en BASIC.

Por lo general utilizamos las siguientes extensiónes para ficheros en una ductora:

extensión	*significado*
ASC	fichero que contiene códigos ASCII
ASM	programa en esamblador
BAK	el llamado 'back up file' (copia de un fichero)
BAS	programa BASIC
COM	el llamado 'utility programma' (programa de utilidad)
TEX	fichero de textos
PIC	el llamado 'picture-file' (fichero de imágenes)

Nota: Está prohibido utilizar para extensiones los siguientes nombres: AUX, CON, LST, PRN y NUL.

Con ciertos comandos queremos, a veces, designar no uno sino un conjunto de programas. En ese caso utilizaremos signos especiales, los llamados comodines ('wild cards').
Así,

```
TEST.*
```

quiere decir: 'todos los programas con el *nombre propio* TEST y una extensión cualquiera'. Si expresamos:

```
*.BAS
```

queremos decir: 'todos los programas con *extensión* BAS'.
Otro ejemplo:

```
TEST*.BAS
```

significa: 'todos los programas con *extensión* BAS, en los cuales los cuatro primeros signos del nombre propio sean TEST'.

Además del signo * que podemos utilizar como wild card, podemos hacer también uso del signo ? asimismo como wild card. Así la designación

```
TEST?
```

hace referencia a todos los ficheros cuyas primeras cuatro letras coinciden con TEST y el quinto carácter puede ser arbitrario. Si disponemos, por ejemplo, de los ficheros TEST1, TEST2, TEST3, entonces TEST? se referirá a todos ellos.

Las instrucciones OPEN, PRINT# y CLOSE

Al trabajar con ficheros, lo primero que debemos hacer es darle un número a este fichero. Esto se hace por medio de la instrucción OPEN. Después podemos llenar el fichero secuencial por medio de instrucción PRINT#.

Hasta este momento esto ha sido una enumeración muy limitado de las instrucciones necesarias para determinar un fichero y para proveerlo de datos.

Vamos ahora a ilustrar detalladamente lo anterior de la mano de un ejemplo.
Ejemplo:

```
10 REM ESCRIBIR EL FICHERO
20 A$="PADRE"
30 B$="MADRE"
40 C$="ABUELO"
50 OPEN "CAS:DATA" FOR OUTPUT AS#1
60 PRINT #1, A$
70 PRINT #1, B$
80 PRINT #1, C$
90 CLOSE #1
```

Las líneas 10 hasta el 40 no presentan ningún problema. En la línea 50 encontramos la instrucción OPEN. Esta tiene la forma:

```
OPEN 'medio de almacenamiento: nombre fichero'
     FOR OUTPUT AS#número del fichero
```

Vemos que el término 'medio de almacenamiento' se rellenó con la palabra lecto-grabadora. Esto implica que probablemente queremos trabajar con la lecto-grabadora como medio de almacenamiento. El nombre del fichero es simplemente DATA. La última parte:

```
FOR OUTPUT AS#1
```

quiere decir: 'vamos a enviar datos desde la memoria del ordenador a la lecto-grabadora' y además 'este fichero lo llamaremos numero 1'.

Notemos que todos los datos que vamos a almacenar así en la lecto-grabadora, van a formar juntos el fichero 'DATA' y que además al referirnos a este fichero, vamos a mencionar su número (#1).

En lugar del término OUTPUT podríamos haber usado el término INPUT. En este caso el ordenador entiende que se van a leer datos del medio de almacenamiento. Luego veremos uno de estos ejemplos.

Las líneas 60 hasta 70 nos dejan ver como almacenar A$, B$ y C$ en el fichero, con ayuda de la instrucción PRINT#. Nótese que inmediatamente detrás de PRINT aparece el número del fichero.

Cuando se hayan leído todos los datos debemos utilizar la instrucción CLOSE. Después de CLOSE vemos el número del fichero. La razón para servirnos de la instrucción CLOSE es la siguiente. El verdadero transporte de datos de la memoria al medio de almacenamiento tiene lugar por intermedio de un bloque auxiliar de memoria (inglés: buffer). Sin ayuda de la instrucción CLOSE podría ser que el bloque auxiliar en su totalidad quedara sin escribir.
Otro ejemplo:

```
10 REM LEER FICHERO
20 OPEN "CAS:DATA" FOR INPUT AS#1
30 INPUT #1, A$, B$, C$
40 PRINT A$, B$, C$
50 CLOSE #1
```

Resultado

```
PADRE   MADRE   ABUELO
```

Vemos que la construcción de este programa es casi igual a la anterior. Si en el anterior aparecía OUTPUT, aquí aparece INPUT. Se trata realmente de leer datos de la lecto-grabadora. En vez de la instrucción PRINT #, vemos ahora INPUT #. Pués hay que leer datos.

La instrucción PRINT de la línea 40 la hemos incluído solamente para asegurarnos de que los datos originales han sido realmente leídos.

Finalmente vamos a dar ejemplos de instrucciones OPEN sin construir un programa completo.

Ejemplo 1

```
OPEN "A:PROG.DAT" FOR INPUT AS#3
```

Es evidente que se trata de un fichero de la unidad A, del cual se van a leer datos. Al fichero se le ha dado el número 3, y el nombre PROG.DAT.

Ejemplo 2

```
OPEN "GRP:TEST" FOR OUTPUT AS#1
```

Esta instrucción se trató en el programa de la introducción. La información del fichero número 1 se expone ahora en la pantalla gráfica.

En el Resumen de instrucciones MSX-BASIC se presenta una detallada descripción de la instrucción OPEN.

Forma en que se almacenan ficheros secuenciales

El nombre secuencial quiere decir que la información se almacena en orden, es decir, una detrás de otra. A continuación vamos a profundizar en los símbolos 'CR', 'LF', ',' que se utilizan como signos de separación. Comenzamos la ilustración de la mano de una explicación de la instrucción PRINT.

Sobre la coma (CR y LF)

Cuando hayamos terminado de ejecutar una instrucción, el ordenador habrá colocado el cursor al *principio* de una *nueva* línea. Para indicar el principio de una línea se usa la abreviatura CR, mientras que LF se refiere al *traslado a otra línea*. CR y LF tienen sus códigos ASCII específicos. Estos signos se utilizan asimismo como signos de separación cuando se colocan datos en cassettes por medio de PRINT #.

Ejemplo 1

```
10 OPEN "CAS:ADDR" FOR OUTPUT AS#1
20 A=15:B=23.5:C=10:D=25.2:E=13
30 PRINT #1, A; B; C
40 PRINT #1, D; E
50 CLOSE #1
```

En este caso, los datos del cassette pueden visualizarse como sigue:

15	,	23.5	,	10	CR	LF	25.2	,	13	CR	LF

Tenga cuidado al colocar literales independientes por medio de la instrucción PRINT #. En éstas debemos introducir siempre una coma como signo de separación.

Ejemplo 2

```
10 OPEN "CAS:NOMBRE" FOR OUTPUT AS#1
20 A$="JOHN":B$="ELLY"
30 PRINT #1, A$;",";B$
40 CLOSE #1
```

En imagen:

JOHN	,	ELLY	CR	LF

Para terminar, haremos las siguientes observaciones:

- La cantidad máxima de ficheros, que pueden estar abiertos simultáneamente, es de 1, a no ser que se haya indicado una cantidad mayor, con el comando MAXFILES (ej. MAXFILES=3) Sólo se pueden utilizar 6 ficheros con la unidad de discos, cuando se usa la instruccion OPEN.

- En lugar de PRINT #, puede usarse la forma PRINT USING #.

- En lugar de INPUT #, puede usarse también la forma LINE INPUT #.

Ficheros de acceso directo (random access files)

Además de los ficheros secuenciales, el MSX conoce también los ficheros de acceso directo. Estos sólo pueden existir cuando se usan unidades de discos. El apéndice D nos enseña una descripción detallada de los mismos.

APÉNDICES

A

USO DE LA LECTO-GRABADORA.

Veamos un breve comentario acerca del uso de la lecto-grabadora. Usemos preferentemente una lecto-grabadora MSX. Otras lecto-grabadoras pueden dar problemas al ajustar el volumen y la tonalidad. Esto lo descubriremos solamente usándolas.

Partimos del siguiente programa de demostración:

```
10 REM DEMOstración de cassette
20 END
```

Para guardar este programa hacemos lo siguiente:

a. conecte la lecto-grabadora
b. asegúrese de que el programa anterior esté presente en la memoria del ordenador.
c. ponga el magnetofón en la posición de 'grabar', y ejecute el comando CSAVE "DEMO".

Al terminar vemos aparecer en la pantalla la palabra OK.

Es posible controlar la grabación utilizando el comando CLOAD? "DEMO", sin olvidarnos del signo de interrogación después del comando CLOAD. Este comando se teclea inmediatamente después de rebobinar la cinta y con la lecto-grabadora en la posición 'PLAY'. El programa original, que va a ser comparado con el de la cinta, se encuentra presente en la memoria del ordenador.

Más tarde, cuando necesitemos leer el programa del cassette, usamos el comando:

```
CLOAD
```

El ordenador leerá el primer programa que encuentre en la cinta.

También podemos teclear un nombre de programa: el comando será entonces: CLOAD 'nombre del programa'. En nuestro ejemplo tenemos:

```
CLOAD "DEMO"
```

- Si pasa algo extraño al leer un programa, hay dos posibilidades:
  – el ordenador comunica 'device I/O error'
  – no aparece ninguna comunicacion.

  Rebobine la cinta y experimente con el volumen y el tono, hasta saber cual es la posición correcta de los mismos antes de leer datos. Generalmente se ajusta el volumen a un 80% y la tonalidad al maximo. A propósito, utilice sólo cintas ferro, y no CrO_2.

- La lecto-grabadora también la podemos utilizar para almacenar ficheros secuenciales.

- Algunas cintas que han sido grabadas de antemano, no las podemos copiar. Esto no significa que la lecto-grabadora funcióne correctamente, sino que la cinta esta protegida.

- Cuando su ordenador MSX tiene una unidad de disco incorporada, o cuando se ha conectado una unidad de disco a traves de una interface antes de cargar un programa, debe pulsar la tecla 'SHIFT' a la vez que enciende su MSX.

Resumen de los comandos

Los siguientes comandos tienen que ver con el manejo de lecto-grabadora. En el Resumen de las instrucciones MSX-BASIC encontramos una descripción detallada de los mismos.

BLOAD	para cargar un programa en código máquina.
BSAVE	para almacenar un programa en código máquina.
CLOAD	para cargar un programa.
CLOSE	para finalizar con la manipulación de un fichero.
CSAVE	para almacenar un programa.
INPUT#	para la lectura de un fichero sequencial.
LINE INPUT#	para la lectura de un fichero sequencial, por línea.
LOAD	para cargar un fichero en forma ASCII.
MERGE	para agregar un programa externo a la memoria.
OPEN	especifica cómo usar un fichero.
PRINT#	sirve para enviar datos, sequencialmente, a un fichero.
PRINT# USING	sirve para enviar datos, sequencialmente, a un fichero, de acuerdo a un formato dado.
SAVE	para almacenar un fichero in forma ASCII.

Resumen de las funciones

Se exponen en el Resumen de las instrucciones MSX-BASIC.

EOF	fin del fichero.
INPUT$#	para cargar datos alfanuméricos.
LINE INPUT$#	para cargar datos alfanuméricos, por línea.
VARPTR#	para encontrar la ubicación del bloque guía de un fichero.

B

USO DE LA IMPRESORA

Podemos conectar al ordenador, tanto impresoras MSX, como impresoras 'no-MSX'. Una impresora MSX es una impresora que ha sido construida especialmente para los sistemas MSX, de forma tal que no tengamos ningún problema al conectarlos al ordenador (conexión, señales de guía, conjunto de caracteres). Con una impresora 'no-MSX' podemos toparnos con esta clase de problemas.

Inicialmente utilizaremos la impresora para imprimir listas de nuestros programas (comando LLIST). También podemos utilizar la impresora para imprimir datos (instrucción LPRINT). Al fin de cuentas, podemos almacenar datos en un fichero secuencial (véase la descripción de MAXFILES en el Resumen de las instrucciones MSX-BASIC).

Resumen de los comandos

Los siguientes comandos tienen aplicación al utilizar la impresora. En el Resumen de las instrucciones MSX-BASIC encontramos una descripción detallada de los mismos.

CLOSE	para terminar ficheros
LLIST	para la impresión de un programa
LFILES	para la impresión de todos los ficheros que se encuentran en el disco.
LPRINT	para la impresión de datos
OPEN	para especificar cómo se va a utilizar un fichero (no podemos leer desde la impresora; sólo podemos usar la forma 'OUTPUT').
PRINT#	para la impresión de datos como si fueran un fichero secuencial.
PRINT# USING	para la impresión secuencial de datos, incluyéndose un formato.

Resumen de las funciones

Se exponen en el Resumen de las instrucciones MSX-BASIC.

VARPTR#	para encontrar la ubicación del bloque guía de un fichero.

C USO DE LOS MANDOS PARA JUEGOS (JOYSTICK)

Estas conexiones estan señaladas en la máquina con 'hand control'. A este conector puede Vd conectarle, una tableta gráfica, lápiz de luz, ratón o bola gráfica.

Resumen de los comandos

Los comandos siguientes tienen que ver con los mandos para juegos (joystick). En el Resumen de las instrucciones MSX-BASIC encontraremos una explicación detallada de los mismos.

ON STRIG GOSUB	controla si el 'botón de disparo' ha sido apretado.
STRIG/ON/OFF/STOP	activa el control del 'botón de disparo' del mando de la palanca para juegos.

Resumen de las funciones

Estas se detallan en el Resumen de las instrucciones MSX-BASIC.

PAD	controla la situación del panel de toque, del 'lápiz de luz' (light pen), del 'ratón' de dibujo (mouse, o 'track ball').
PDL	controla la situación de la palanca.
STICK	controla la situación del mando de la palanca para juegos.
STRIG	nos dice si el botón de fuego ha sido apretado o no.

D USO DE DISCOS FLEXIBLES

Si disponemos de una ductora de discos, podemos utilizarla para almacenar programas o datos. En muchos casos se venden programas en disco flexible. Al introducir el disco en la ductora, por lo general, empieza automáticamente un programa al poner en marcha el ordenador. Esto depende de si en el disco se ha incluido o no un programa cuyo nombre es AUTOEXEC.BAS, sobre el cual diremos algo más adelante.

Si no queremos que el ordenador comience con un programa automáticamente, primero ponemos en marcha el ordenador, y cuando aparezca 'Ok', introducimos el disco en la unidad. Por medio del comando FILES obtenemos una lista de todos los ficheros presentes en el disco. Este tópico lo trataremos también más tarde.

A continuación profundizamos detalladamente en el uso de los discos flexibles.

Tratamos el comando COPY (para copiar un fichero), FILES (para obtener una lista de los ficheros presentes), y KILL (para borrar de un fichero).

Luego trataremos el comando FORMAT (para preparar un disco nuevo para su uso), SAVE (para escribir un fichero en el disco) y LOAD (para leer un fichero en la memoria).

Finalmente explicamos como utilizar los llamados ficheros de acceso directo (random access files). Concluimos este apendice con un resumen de los funciones/comandos que tienen que ver con el manejo de discos.

Algo acerca de la unidad de gestion de perifericos y los comandos COPY, FILES y KILL

El lugar donde puede colocarse un disquete se designa generalmente con el nombre de 'ductora' (en ingles driver es el con-ductor). Podemos acoplar más de una ductora a nuestro ordenador, y en base a este hecho, puede uno preguntarse '¿cómo indicaremos a qué ductora nos referimos?' La respuesta es muy sencilla: indicamos las ductoras simbólicamente por medio de letras. Si tenemos, por ejemplo, dos ductoras, indicaremos éstras con las letras A y B. Estas letras las utilizamos para indicar en qué unidad ductora se encuentra un fichero determinado.
Así,

```
A:TEST.BAS
```

significa: 'el programa TEST.BAS que se encuentra en la unidad A'.

Daremos ahora un ejemplo de un comando en el que aparecen tanto indicaciones de ficheros como de ductoras, éste es el comando COPY.
Sirve para hacer copias de ficheros.

El comando:

```
COPY "A:TEST.BAS" TO "B:"
```

significa: 'copia el programa TEST.BAS que se encuentra en el disquete de la ductora A hacia el disquete B'.
Otro ejemplo:

```
COPY "A:TEST.BAS" TO "B:EXP1.BAS"
```

significa: 'copia el programa TEST.BAS y almacénalo bajo el nombre EXP1.BAS en el disquete que se encuenta en la ductora B'.

Otro ejemplo que, en este caso, tiene que ver con el comando que da las listas de los ficheros que se encuentran en un disquete. Este comando lleva el nombre de FILES.
Así,

```
FILES "A:"
```

quiere decir: 'da una lista de todos los ficheros que se encuentran en el disquete de la ductora A'.

También podemos dar algún nombre en el comando FILES, para ver si ese fichero se encuentra en el disquete, p.ej.

```
FILES "A:TEST.BAS"
```

Si ese fichero se encuentra allí, aparecerá el nombre TEST.BAS una vez más en la pantalla. Si éste no es el caso, entonces el ordenador anunciará:

```
File not found
```

lo cual quiere decir que no ha encontrado el citado fichero.

Ahora un ejemplo para ver cómo se borran los ficheros. Para ello usaremos el comando KILL.
Así,

```
KILL "A:TEST.BAS"
```

significa: 'borra el programa TEST.BAS que se encuentra en el disquete de la ductora A'.

¿Qué querrá decir, entonces, lo siguiente?

```
KILL "A:*.*"
```

...de acuerdo, borra todos los programas que se encuentran en el disquete de la ductora A.

Veremos, a continuación, cómo tenemos que utilizar los nombres A y B si sólo disponemos de una unidad ductora.

AUTOEXEC.BAS Si un fichero tiene el nombre AUTOEXEC.BAS entonces este comenzará automaticamente al connectar el sistema, siempre y cuando este se *encuentre presente en la ductora A.*

FORMAT Un disco que todavía no ha sido usado debe sufrir un tratamiento que llamamos formatear. Este tratamiento significa que el disco debe ser preparado para que luego pueden almacenarse ficheros en él.
Para ello tecleamos:

```
_FORMAT
```

o

```
CALL FORMAT
```

Cuando el disco esté formateado, aparecerá en la pantalla:

```
Format complete
Ok
```

Para comprobar si el disco esta y a preparado para recibir un programa, escibiremos un pegueño programa de demostración:

```
10 PRINT "MSX DISK BASIC"
```

Para almacenar este programa en el disco, haremos uso del comando SAVE. Damos el siguiente comando:

```
SAVE "A:TEST.BAS"
```

que quiere decir: 'guarde el programa llamado TEST.BAS que se encuentra en el disco de la unidad A'.

Después de pulsar la tecla RETURN, se oye un zumbido... ¿estará el programa almacenado en el disco?

Esto lo podemos comprobar borrando con el comando NEW el programa que se encuentra todavía en la memoria (¡hágalo!). Ahora cargamos el programa del disco por medio del llamado comando LOAD:

```
LOAD "A:TEST.BAS"
```

Después de haber pulsado la tecla RETURN, volverá a oírse el zumbido.

Si cuando se acabe damos el comando LIST, veremos que nuestro programa se encuentra en la memoria del ordenador.

Para comprobar si el fichero TEST.BAS se encontraba realmente en el disco de la unidad A, podíamos haber dado también el comando FILES que acabamos de explicar, p. ej.:

```
FILES A:*.*
```

(¡hágalo!), o más breve:

```
FILES
```

Solo disponemos de una unidad ductora

En la mayoría de los casos sólo se dispondrá de una ductora. Entonces ¿qué hacer?

Pues bien, en este caso, los nombres A y B de las unidades de discos se refieren no a las ductoras sino de los discos. Ilustraremos esto con el ejemplo siguiente:

Imagine que tenemos dos discos (A y B) y que queremos copiar el programa TEST.BAS desde el disco A al B. Sunponga también que el disco A se encuentra todavía en la unidad. Damos el comando siguiente:

```
COPY "A:TEST.BAS" TO "B:"
```

Ahora, este programa será cargado primero en la memoria del ordenador y, a continuación, el ordenador visualizará:

```
Insert diskette for drive B
and strike a key when ready
```

En este caso, tendremos que meter el disco B en la unidad.

Si el programa no puede ser copiado de una vez, habrá que hacerlo por etapas. el ordenador indicará cada vez si debemos insertar el disco A o el B en la unidad.

Consejo práctico: anote en el disco el nombre, es decir si se trata de A o B, esto le evitará muchos problemas.

Ficheros secuenciales

El MSX Disk BASIC ofrece también la posiblidad de trabajar con archivos secuenciales. El principio de un fichero secuencial ya lo hemos explicado de modo que ahora ya no nos pareremos en ello.

Pero hay un punto que exige que se le preste mayor atención, se trata de la función APPEND. Por medio de esta función podemos ampliar un archivo secuencial, cosa que no es posible en un cassette.

Si queremos almacenar más datos en un fichero existente, o sea, si queremos ampliar el fichero, daremos la siguiente instrucción OPEN:

```
OPEN "nombre" FOR APPEND AS#número
```

Suponga que hemos hecho un pegueño fichero con el programa que sigue:

```
10 OPEN "A:DAT" FOR OUTPUT AS#1
20 INPUT X$
30 IF X$="END" THEN CLOSE #1:END
40 PRINT #1, X$
50 GOTO 20
```

Cuando hayamos almacenado algunos literales en A:DAT, se encontrará en el disco el fichero A:DAT.

El programa que damos a continuación muestra cómo podemos añadir más datos al fichero anterior:

```
10 OPEN "A:DAT" FOR APPEND AS #1
20 INPUT X$
30 IF X$="END" THEN CLOSE #1:END
40 PRINT #1, X$
50 GOTO 20
```

La diferencia con el programa anterior no es más que la palabra APPEND.

Ficheros de acceso directo

Un 'random access file' es un archivo de unas características especiales, es decir que, dentro de ciertas posiciones del fichero, podemos almacenar o sacar datos. Para mejor comprensión, miremos la siguiente figura:



En esta figura vemos una sucesión de registros que, a su vez, están divididos en campos (FIELDS).

Esto significa que cada registro tiene una longitud fija (LEN) en términos de bytes y esta longitud está compuesta a su vez de campos que tienen una longitud fija. Para definir un fichero directo así, tenemos que indicar primero LEN, p.ej.:

```
OPEN "A:DATA" AS #1 LEN=34
```

En este ejemplo, cada registro tiene una longitud de 34 bytes. Como en esta instrucción OPEN no incluimos la expresión FOR INPUT, OUTPUT o APPEND, el ordenador sabe automáticamente que se trata de un fichero directo.

Además tenemos que especificar de qué está compuesto cada campo y asimismo, con qué nombre simbólico lo vamos a designar.
Por ejemplo:

```
FIELD #1, 20 AS A$, 8 AS D$, 4 AS S$, 2 AS I$
```

Aquí vemos que cada registro consta de 4 campos denominados A$, D$, S$ e IS, que contienen la cantidad indicada de bytes. La cantidad total de bytes, naturalmente, tiene que ser igual a la cantidad que hemos indicado en LEN (controle si es así).

Ahora hemos fijado, como si dijeramos, el armazón del fichero y podemos mirar cómo vamos a colocar los datos en él. Para esto nos serviremos de las instrucciones LSET y RSET.

El ejemplo siguiente muestra las diferentes posibilidades que hay:

```
LSET A$=WA$
RSET D$=MKD$(WD#)
RSET S$=MKS$(WS!)
RSET I$=MKI$(WI%)
```

Aquí WA$ representa un literal WD# un número de doble precisión, WS! un número de precisión sencilla y WI% un número entero. Observe que cada número tiene que ser primero convertido, siempre, en literal por medio de la función adecuada.

Ahora tenemos que colocar estos valores en el fichero... pero ¿cómo? Pues bien, esto depende de dónde, es decir, en qué registro queremos almacenarlos. Para ello nos serviremos de la instrucción

```
PUT #1,número de registro
```

P. ej.:

```
PUT #1,3
```

Observe que después de esto son almacenados en el fichero todos los datos que hemos asignado a A$, D$ e I$ por medio de LSET y RSET, de modo que podemos asignar de nuevo, del mismo modo, otros datos a A$, D$, S$ e I$ que, a su vez, podemos almacenar en otro registro.

Finalmente, al final de un programa debemos siempre cerrar el fichero, o sea:

```
CLOSE #1
```

Para la salida, definiremos el fichero directo del mismo modo que acabamos de explicar.

Si en un programa tenemos tanto acciones de entrada como de salida en un fichero, sólo tenemos que definir el fichero, del modo conocido, una vez.
La salida se realiza por medio de la instrucción GET.
Así, por medio de:

```
GET #1,3
```

será asignado el contenido de los campos del registro 3 a A$, D$, S$ e I$. Estos valores los podemos imprimir, por ejemplo, por medio de

```
PRINT A$
PRINT CVD(D$)
PRINT CVS(S$)
PRINT CVI(I$)
```

Fíjese en las funciones CVD, CVS y CVI véase el Resumen de las instrucciones MSX-BASIC.

Resumen de los comandos

Los siguiente comandos tienen que ver con el manejo de discos. En el Resumen de las instrucciones MSX-BASIC encontramos una descripción detallada de los mismos.

BLOAD	para cargar un programa en código de máquina o para cargar una imagen almacenada en un medio externo.
BSAVE	para almacenar un programa en código máquina o una imagen.
CLOSE	para finalizar con la manipulación de un fichero.
COPY	para copiar ficheros o una parte de la pantalla.
DSKO$	para escribir datos en un sector específico del disco.
FIELD	para permitirnos la entrada al bloque de datos que controla un fichero de acceso directo.
FILES	nos proporciona una lista de los ficheros del disco.
LFILES	igual que FILES, pero la lista será impresa.
_FORMAT	para el formateado de un disco.
GET	para leer un dato en el bloque de control de un fichero de acceso directo.
INPUT#	para la lectura de un fichero secuencial.
KILL	para borrar de un fichero.
LINE INPUT#	para la lectura de un fichero secuencial, por línea.
LOAD	para cargar un fichero.
LSET, RSET	se utiliza con ficheros de acceso directo.
MAXFILES	da el número máximo de ficheros que pueden estar abiertos simultáneamente. Sólo puede haber 6 ficheros abiertos en un programa.
MERGE	para agregar un programa externo a la memoria.
NAME	para darle un nuevo nombre a un fichero.
OPEN	especifica cómo usar un fichero.
PRINT#	sirve para enviar datos, secuencialmente, a un fichero.
PRINT# USING	sirve para enviar datos, secuencialmente, a un fichero, de acuerdo a un formato dado.
PUT	se utiliza con ficheros de acceso directo.
SAVE	para almacenar un fichero en disco.
_SYSTEM	para regresar al sistema MSX-DOS.

Resumen de las funciones

Éstas se detallan en el Resumen de las instrucciones MSX-BASIC.

CVI, CVS, CVD	para la conversión de cadenas (strings) en números. Se utilizan con ficheros de acceso directo.
DSKF	da el espacio libre que queda en disco.
DSKI$	para cargar un sector.
EOF	fin del fichero.
INPUT$#	para cargar datos alfanuméricos.
LINE INPUT$#	para cargar datos alfanuméricos, por línea.
LOC	da el último sector de un fichero.
LOF	da la longitud de un fichero.

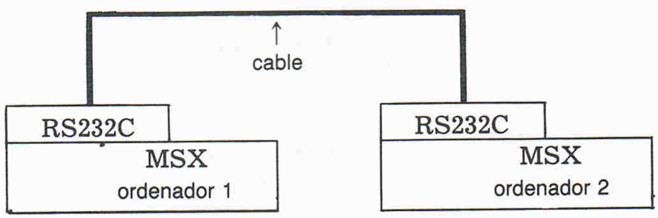
MKI$, MKS$, MKD$	véase CVI, CVS, CVD. Ahora se trata de convertir un número en una cadena.
VARPTR#	para encontrar la ubicación del bloque guía de un fichero.

E

USO DE LA INTERFACE RS232C

El concepto de comunicación es importante en el mundo de la informática. Con él nos referimos al envío de información de un lugar a otro. Imagine, por ejemplo, dos ordenadores conectados entre sí. ¿No es algo maravilloso poder intercambiar programas, o ficheros de datos de una manera sencilla? Pues, bien, para poder conseguir esto ha sido desarrollada la llamada interface RS232C para el ordenador MSX. Este aparato sirve para transferir en un formato adecuado la información que quiera enviarse a otro dispositivo. En lo que se refiere a laRS232C, nos referimos a una interface serie, ya que los ceros y unos en que, en definitiva están representados todos los datos, son enviados aquí unos detrás de los otros, es decir, en serie (o secuencialmente).

Para conseguir que este proceso se desarrolle como es debido, se necesitan, como es natural, señales de control. A continuación mostraremos cómo pueden comunicarse entre sí dos ordenadores MSX, equipados con una interface RS232C cada uno, o sea, en imagen:



En nuestro ejemplo se comunican entre sí dos ordenadores MSX; pero, en general, el segundo ordenador podría ser cualquier otro aparato que incorpore una interface RS232C (piénsese por ejemplo en un modem para comunicar con otro ordenador a través del teléfono).

La interface misma consiste en un cartucho sin cable. En este cartucho se encuentra un enchufe ancho (llamado 25-pins Centronics connector). En el mercado existen cables que se pueden comprar sueltos, que pueden servir para conectar entre sí los cartuchos de los dos ordenadores (véase la figura).

Enviar un programa

Supongamos que escribimos el programa siguiente en el ordenador 1:

```
10 PRINT "RS232C"
20 PRINT "MSX"
```

A continuación damos el comando siguiente:

```
SAVE "COM:"
```

Queremos, como quien dice, 'salvar' este programa pr medio de la indicación COM en que COM se refiere a nuestra interface RS232C. Si ahora escribimos en el ordenador 2:

```
LOAD "COM:"
```

Será trasmitido el programa cuando pulsemos la tecla return de los dos ordenadores. Observe que si esperamos desmasiado, después de haber dado el comando SAVE, para escribir el comando LOAD, se producirá una señal de error (Device I/O error). La razón es que, al pulsar SAVE, se hizo saber que el ordenador 1 quería enviar datos y, entonces, investiga si el ordenador 2 está dispuesto a recibirlos (dando el comando LOAD). Si esto tarda demasiado, aparecerá la citada señal de error.

De hecho ésto ocurre en toda clase de comunicaciones: si el aparato receptor no indica que está listo para la recepción de datos, la comunicación se interrumpe automáticamente.

En lugar de LOAD 'COM:', podíamos haber dado también RUN 'COM:', en este caso se ejecutaría el programa inmediatamente.

Enviar datos

El ejemplo que damos a continuación muestra cómo transmitimos datos en forma de ficheros del ordenador 1 al ordenador 2. En el ordenador 1, escribimos el siguiente programa:

```
10 OPEN "COM:" FOR OUTPUT AS#1
20 INPUT A$
30 PRINT #1, A$
40 CLOSE #1
```

En el ordenador 2, escribimos el siguiente programa:

```
10 OPEN "COM:" FOR INPUT AS#1
20 INPUT #1, A$
30 PRINT A$
40 CLOSE #1
```

Si, después de poner en marcha el programa, escribimos en el ordenador 1 el literal MSX, aparecerá en la pantalla del ordenador 2 (si es que en el programa del ordenador 2 hemos dado también el comando RUN MSX)
Otro ejemplo:
Si damos en los dos ordenadores el comando

```
CALL COMTERM
```

todo lo que se escriba en el ordenador 1, será visualizado también en el ordenador 2, y viceversa, todo lo escrito en 2 aparecerá también en 1. En este caso habremos hecho de los dos ordenadores una 'terminal tonta' (tipo de aparato para tasmitir información), pudiendo intercambiar información de una manera sencilla.

Conexion a un aparato que no sea un ordenador MSX (como un ordenador de otra marca)

Una conexión de éste tipo requiere la previa definición de parámetros como:
- la velocidad de la transmisión de información
- el llamado control de paridad
- la longitud de los bytes
- la cantidad de los llamados bits de parada

Todo lo auteriormente tratado en este apartado nos pone de manifiesto que sólo si dispone de unas instrucciones claras de cómo hay que establecer la comunicación, podrá ser ejecutable por un principiante.

Resumen de los comandos

Los siguientes comandos tienen relación con el manejo de la interface RS232C. Una descripción detallada la encontramos en el Resumen de las instrucciones MSX-BASIC.

CALL COMINI	para poner a punto el interface RS232C. (Velocidad en baudios, longitud de los 'bloques' que se envían, etc.)
CALL COMON CALL COMOFF CALL COMSTOP CALL COM	estas instrucciones permiten ejecutar una subrutina BASIC tan pronto como se reciba un signo a través de la interface RS232C.
CALL COMTERM	comienza con la llamada 'modo emulador de terminal' (terminal emulator mode).
CALL COMBREAK	para el envío de los llamados caracteres de interrupción.
CALL COMDTR	para indicar que desconectamos temporalmente nuestra interface.
CALL COMSTAT	si se produce un error podemos enterarnos de qué es lo que ha fallado.
CLOSE	para finalizar la comunicación mediante la interface RS232C.
INPUT#	para recibir datos mediante la interface.
LOAD	para recibir un programa mediante la interface.
LINE INPUT#	para recibir datos por medio de la interface, por línea.
MERGE	para intercalar un programa por medio de la interface.
OPEN	para iniciar una comunicación via interface.
PRINT#	para enviar datos via interface.
PRINT# USING	para enviar datos via interface, de acuerdo a un formato especificado.
SAVE	para enviar un programa via interface.

Resumen de las funciones

Estas se decriben con detalle en el Resumen de instrucciones MSX-BASIC.

EOF	fin del fichero enviado por la interface.
INPUT$#	para recibir datos alfanuméricos por la interface.
LINE INPUT$#	para recibir datos alfanuméricos por la interface, por línea.
VARPTR#	para hallar la ubicación (dirección) del bloque guia de ficheros.

F SIGNOS Y EXPRESIONES ESPECIALES

Signos operacionales

La tabla de abajo da un cuadro de los signos que pueden ser usados para operaciones matemáticas. La última cifra da el valor de prioridad: las expresiones se resuelven con arreglo a estos valores de prioridad (desde la cifra más baja hasta la más alta). Si hay valores de prioridad iguales, se resolverá la expresión de izquierda a derecha.

signo	*significado*	*ejemplo*	*prioridad*
+	sumar	3+5	6
-	restar	5-3	6
*	multiplicar	5x3	3
/	dividir	5/3	3
^	elevación a potencia	2^3	1
-	signo cambiado	-3	4
MOD	resto módulo	12 MOD 3	5

Signos y expresiones logicas

Los siguientes símbolos pueden aparecer en expresiones en las que se investigará si son ciertos o no (p.ej. después del término IF)

signo	*significado*	*ejemplo*
=	es igual a	IF A = B THEN...
<	menor que	IF A < B THEN...
>	mayor que	IF A > THEN...
<> o ><	distinto de	IF A <> B THEN...
<= o =<	menor que o igual a	IF A < = B THEN...
>= o =>	mayor que o igual a	IF A > = B THEN...

Terminos para combinar expresiones logicas

Expresiones como A=B y C<>D pueden combinase con ayuda de términos específicos.

La tabla siguiente muestra los diferentes términos con las llamadas tablas de verdad. En ellas, 1 indica que la expresión es 'cierta' y 0 que no es 'cierta' (es errónea). Así se ve en el término AND, por ejemplo:

U1 **AND** U2

Si 1 U1 y U2 son iguales a 1. U1 y U2 significan expresiones co-

mo A=B y C<>D. La tabla, en este caso, indica que la expresión completa es cierta (1) si U1 y U2 también son ciertas (1).

término			*tabla de certeza*
NOT	U1		NOT U1
(negación)	1		0
	0		1
AND	U1	U2	U1 AND U2
(producto lógico)	1	1	1
	1	0	0
	0	1	0
	0	0	0
OR	U1	U2	U1 OR U2
(suma lógico)	1	1	1
	1	0	1
	0	1	1
	0	0	0
XOR	U1	U2	U1 XOR U2
(o exclusivo)	1	1	0
	1	0	1
	0	1	1
	0	0	0
EQV	U1	U2	U1 EQV U2
	1	1	1
	1	0	0
	0	1	0
	0	0	1
IMP	U1	U2	U1 IMP U2
(implicación lógica)	1	1	1
	1	0	0
	0	1	1
	0	0	1

Signos operacionales en literales

En las literales sólo puede usarse el signo +, para acoplar dos literales entre sí. Ejemplo:

"AB" + "CD" da "ABCD"

G

MENSAJES DE ERROR

Bad allocation table (60)
El disquete no está inicializado (no formateado o erróneamente formateado).

Bad drive name (62)
Es señalada una unidad de disquete que no exite.

Bad file mode (61)
Se hace uso de PUT, GET o EOF con un fichero secuencial o se trata de cargar por medio de LOAD un fichero aleatorio. Finalmente, esta señal de error puede producirse si se usa la instrucción OPEN en un fichero que no está definido correctamente.

BAD file name (56)
El nombre del fichero es erróneo. el nombre del dispositivo ('device') en un comando OPEN, SAVE o LOAD es erróneo.

BAD file number (52)
El número de un fichero es mayor de lo permitido o se indica un fichero que todavía no está 'abierto'.

Can't CONTINUE (17)
Se trata de dar el comando CONT cuando el programa no ha sido interrumpido por un error o una instrucción STOP. También se produce al modificar un programa probando CONT inmediatamente después.

Device I/O error (19)
La carga de un programa o archivo no marcha bien. ¿Está bien conectada la lectógrabadora de datos? ¿Se ha indicado el dispositivo de entrada/salida correcto?

Direct statement in file (57)
En el programa que se está cargando aparece una instrucción sin número de línea o el fichero que se carga no es un programa.

Disk full (66)
Ha sido utilizada toda la capacidad del disquete.

Disk I/O error (690)
Se ha constatado un error en la acción de entrada o salida. Este es lo que se llama un 'error fatal', que quiere decir que el sistema

MSX Disk BASIC tiene que volver a ser puesto en marcha por el usuario.

Disk offline (70)
No hay disquete en la unidad indicada.

Disk write protected (68)
Se trata de almacenar algo en un disquete 'write protected' (es decir, un disquete a prueba de escritura).

Division by zero (11)
No está permitido devidir por 0. Este error puede producise también si se devide por una variable a la que no se ha asignado un valor de antemano.

Field overflow (50)
El número de bytes asignado a la instrucción FIELD es mayor de 256.

File already exists (65)
El nuevo nombre del fichero (por medio del comando NAME) es ya el nombre de otro fichero que se encuentra en el disquete.

File already open (54)
Se trata de abrir un fichero que ya está abierto.

File not found (53)
Un comando LOAD, KILL o una instrucción OPEN refiere a un fichero que no se encuentra en el disquete.

File not OPEN (59)
Se indica un fichero que todavía no ha sido abierto por una instrucción OPEN.

File still open (64)
No se ha cerrado el fichero.

Illegal direct (12)
Se trata de ejecutar una instrucción como si fuera un comando mientras que eso no está permitido en esta instrucción.

Illegal function call (5)
Se llama una función BASIC con un argumento erróneo. Esta señal se produce en:

1 índice de matriz negativo o en índices que caen fuera del alcance de la matriz;
2 un argumento negativo en LOG o SQR;
3 una mantisa negativa con un exponente real;
4 el uso de la función USR sin haber especificado una dirección de comienzo;
5 un argumento erróneo de MID$, LEFTR$, RIGHT$, INP, OUT, WAIT, PEEK, POKE, TAB, SPC, STRING$, SPACE$, INSTR, ON...GOTO.
6 la ejecución de un comando gráfico siendo así que la pantalla no se encuentra en el modo gráfico.

Input past end (55)
Se trata de introducir un dato cuando ya han sido introducidos todos los datos. También puede producise si el fichero está vacío, es decir, no contiene ningún dato.

Internal error (51)
Hay un error en el intérprete de BASIC mismo, por ejemplo, algún deterioro de la memoria ROM.

Line buffer overflow (25)
La memoria de almacenamiento intermedio para la introducción de una línea está llena.

Missing operand (24)
Algo no marcha bien con respecto a uno de los datos que hay que utilizar en una expresión.

NEXT without FOR (1)
Ha sido encontrada una instrucción NEXT sin la instrucción FOR correspondiente.

NO RESUME (21)
La rutina de correción de errores utilizada no contiene la instrucción RESUME.

OUT of DATA (4)
Con la instrucción READ se intentan leer datos que no han sido fijados en una lista de DATA.

Out of memory (7)
El programa es demasiado grande para el espacio de memoria disponible o el programa contiene demasiadas variables, GOSUBs o bucles FOR.

Out of string space (14)
El espacio de memoria destinado a literales está lleno. Con la instrucción CLEAR puede ampliarse este espacio.

Overflow (6)
Un cálculo produce un número demasiado grande (el llamado overflow). también puede ser que un valor especificado esté fuera del alcance permitido.

Redimensioned array (10)
Se intenta especificar dos veces la misma variable por medio de DIM.
También puede producirse cuando una instrucción DIM sigue a otra instrucción en la que ya aparece una variable de matriz.

Rename across disk (71)
Con el comando NAME se asigna un nuevo nombre de fichero en el cual la indicación de la trasmisión es distinta de la trasmisión en la que se encuentra el disquete con el fichero referido.

RETURN without GOSUB (3)
Ha sido encontrada una instrucción RETURN sin que se haya saltado a una subrutina a través de GOSUB.

RESUME without error (22)
Ha sido encontrada una instrucción RESUME sin la correspondiente rutina de corrección de errores.

Sequential I/O only (58)
Se quiere leer un fichero secuencial como si fuera un fichero de acceso directo.

String formula too complex (16)
La expresión literal es demasiado complicada y, por lo tanto, debe ser dividida en trozos más pequeños.

String too long (15)
El literal consta de más de 255 signos.

Subscript out of range (9)
Se utilizan números de índice de matriz que estan fuera del alcance especificado.

Syntax error (2)
Indicación general de error: la expresión no se adapta a las reglas de BASIC.

Too many files (67)
Se intenta crear un fichero nuevo siendo que ya hay 255 ficheros.

Type mismatch (13)
Se intenta asignar a una variable literal un valor numérico, o al revés.

Undefined line number (8)
Tiene lugar el envío a un número de línea no existente.

Undefined user function (18)
Ha sido llamada una función FN que no ha sido definida de antemano con ayuda de una instrucción DEF FN.

Unprintable error (23, 26-49, 60-255)
Para este código no existe una señal de error estándar.

Verify error (20)
Se constata una diferencia entre el programa grabado y el programa que se encuentra en la memoria.

H

SECUENCIAS DE ESCAPE

Entendemos como secuencias de escape una serie de códigos utilizados en la pantalla para indicar una acción determinada. Para indicar la secuencia de escape <ESC>B tecleamos:

```
PRINT CHR$(27)+"B"
```

La tabla siguiente nos enseña las diferentes posibilidades:

<ESC> A	cursor una línea hacia arriba
<ESC> B	cursor una línea hacia abajo
<ESC> C	cursor una posición hacia la derecha
<ESC> D	cursor una posición hacia la izquierda
<ESC> H	cursor hacia la esquina superior izquierda
<ESC> Y	<número de fila+32> <número de columna+32> cursor a una posición determinada
<ESC> j	CLS
<ESC> E	CLS
<ESC> K	borrar hasta el fin del renglón
<ESC> J	borrar hasta el final de la pantalla
<ESC> I	borrar la línea en su totalidad
<ESC> L	insertar una línea
<ESC> M	eliminar una línea
<ESC> x4	cambie el cursor en un cuadrado
<ESC> x5	borrar el cursor
<ESC> y4	cambie el cursor en una rayita
<ESC> y5	hace aparecer nuevamente el cursor

I CODIGOS DE CONTROL

Algunas teclas tienen una función especial en combinación con la tecla CTRL. Así, si se pulsa la tecla CTRL y, a continuación, con ésta pulsada se pulsa L, será borrada la pantalla. Esta acción se indica con CTRL/L. De este modo CTRL/R quiere decir: pulsar R mientras se mantie apretada CTRL. El cuadro siguiente muestra todas las posibilidades.

CTRL/A	la tecla que se pulse a continuación llama a un símbolo de la serie alternativa de símbolos.
CTRL/B	traslada el cursor a la palabra anterior.
CTRL/C	para el comando AUTO.
CTRL/E	serán borrados todos los símbolos a partir del cursor.
CTRL/F	traslada el cursor a la palabra siguiente.
CTRL/G	BEEP.
CTRL/H	coincide con la tecla BS.
CTRL/I	coincide con TAB: a la siguiente parada.
CTRL/J	traslada el cursor al comienzo dela línea siguiente.
CTRL/K	traslada el cursor al ángulo superior izquierdo.
CTRL/L	coincide con CLS.
CTRL/M	coincide con la tecla RETURN.
CTRL/N	traslada el cursor al final de la línea (es decir, al primer puesto después del último símbolo de dicha línea).
CTRL/R	coincide con la tecla INS.
CTRL/U	borra la línea.
CTRL/\	coincide con la tecla del cursor →.
CTRL/]	coincide con la tecla del cursor ←.
CTRL/∧	coincide con la tecla del cursor ↑.
CTRL/–	coincide con la tecla del cursor ↓.

J

CONFECCION DE SIMBOLOS

En este apéndice se da una panorámica de todos los símbolos (caracteres) y sus códigos correspondientes.

Estos pueden ser utilizados, por ejemplo, en la función CHR$. Así, el signo A se puede obtener por PRINT "A" pero también por medio de PRINT CHR$(65).

En el modo SCREEN 0 las dos columnas de la derecha de la matriz, con la cual se define cada uno de los símbolos, no están representadas.

Puede verse la diferencia comparando

```
10 SCREEN 0
20 PRINT CHR$(210)
```

y

```
10 SCREEN 1
20 PRINT CHR$(210)
```

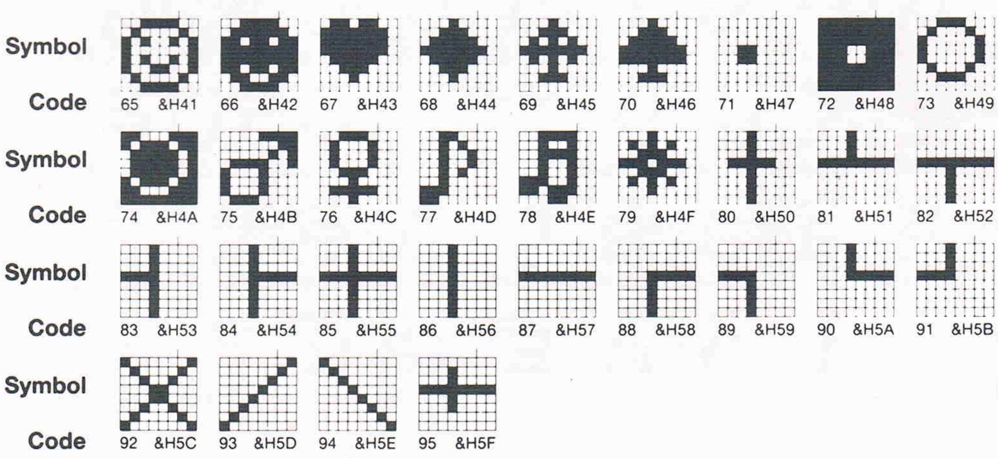
En las páginas siguientes vemos dibujados los símbolos estandar.

Símbolos alternativos

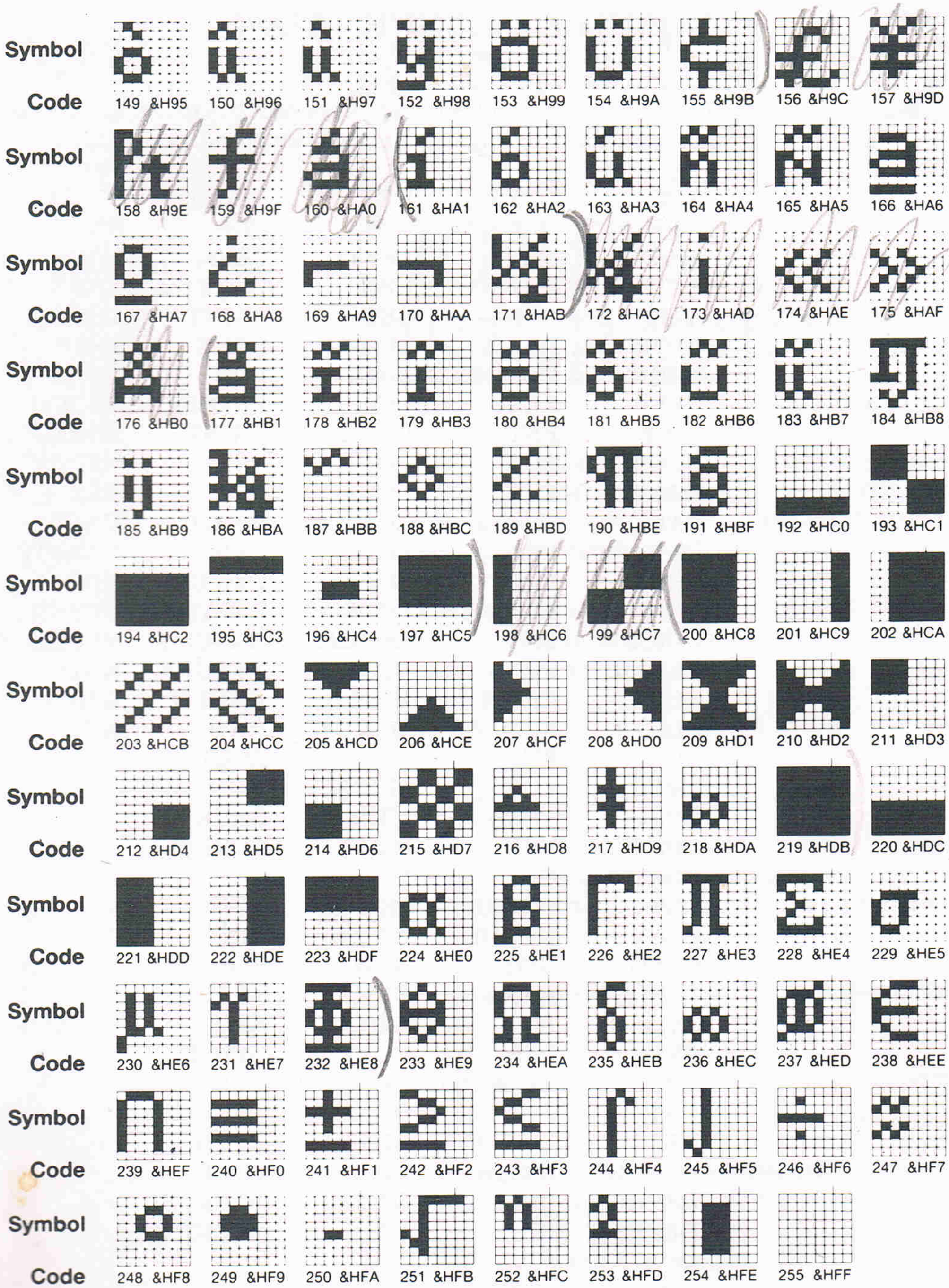
Además de los símbolos vistos, podemos también obtener los siguientes por medio de la inclusión de CHR$ en la instrucción PRINT. Por ejemplo:

```
10 SCREEN 1
20 PRINT CHR$(65)
30 PRINT CHR$(1) + CHR$(65)
```

El renglón 20 genera una A y el 30 da como resultado la representación de una cara.



Symbol		!	"	#	$	%	&	'	(
Code	32 &H20	33 &H21	34 &H22	35 &H23	36 &H24	37 &H25	38 &H26	39 &H27	40 &H28
Symbol	)	*	+	,	-	.	/	0	1
Code	41 &H29	42 &H2A	43 &H2B	44 &H2C	45 &H2D	46 &H2E	47 &H2F	48 &H30	49 &H31
Symbol	2	3	4	5	6	7	8	9	:
Code	50 &H32	51 &H33	52 &H34	53 &H35	54 &H36	55 &H37	56 &H38	57 &H39	58 &H3A
Symbol	;	<	=	>	?	@	A	B	C
Code	59 &H3B	60 &H3C	61 &H3D	62 &H3E	63 &H3F	64 &H40	65 &H41	66 &H42	67 &H43
Symbol	D	E	F	G	H	I	J	K	L
Code	68 &H44	69 &H45	70 &H46	71 &H47	72 &H48	73 &H49	74 &H4A	75 &H4B	76 &H4C
Symbol	M	N	O	P	Q	R	S	T	U
Code	77 &H4D	78 &H4E	79 &H4F	80 &H50	81 &H51	82 &H52	83 &H53	84 &H54	85 &H55
Symbol	V	W	X	Y	Z	[	\	]	^
Code	86 &H56	87 &H57	88 &H58	89 &H59	90 &H5A	91 &H5B	92 &H5C	93 &H5D	94 &H5E
Symbol	_	`	a	b	c	d	e	f	g
Code	95 &H5F	96 &H60	97 &H61	98 &H62	99 &H63	100 &H64	101 &H65	102 &H66	103 &H67
Symbol	h	i	j	k	l	m	n	o	p
Code	104 &H68	105 &H69	106 &H6A	107 &H6B	108 &H6C	109 &H6D	110 &H6E	111 &H6F	112 &H70
Symbol	q	r	s	t	u	v	w	x	y
Code	113 &H71	114 &H72	115 &H73	116 &H74	117 &H75	118 &H76	119 &H77	120 &H78	121 &H79
Symbol	z	{	|	}	~		Ç	ü	é
Code	122 &H7A	123 &H7B	124 &H7C	125 &H7D	126 &H7E	127 &H7F	128 &H80	129 &H81	130 &H82
Symbol	â	ä	à	å	ç	ê	ë	è	ï
Code	131 &H83	132 &H84	133 &H85	134 &H86	135 &H87	136 &H88	137 &H89	138 &H8A	139 &H8B
Symbol	î	ì	Ä	Å	É	æ	Æ	ô	ö
Code	140 &H8C	141 &H8D	142 &H8E	143 &H8F	144 &H90	145 &H91	146 &H92	147 &H93	148 &H94


Symbol
Code
149 &H95
150 &H96
151 &H97
152 &H98
153 &H99
154 &H9A
155 &H9B
156 &H9C
157 &H9D
158 &H9E
159 &H9F
160 &HA0
161 &HA1
162 &HA2
163 &HA3
164 &HA4
165 &HA5
166 &HA6
167 &HA7
168 &HA8
169 &HA9
170 &HAA
171 &HAB
172 &HAC
173 &HAD
174 &HAE
175 &HAF
176 &HB0
177 &HB1
178 &HB2
179 &HB3
180 &HB4
181 &HB5
182 &HB6
183 &HB7
184 &HB8
185 &HB9
186 &HBA
187 &HBB
188 &HBC
189 &HBD
190 &HBE
191 &HBF
192 &HC0
193 &HC1
194 &HC2
195 &HC3
196 &HC4
197 &HC5
198 &HC6
199 &HC7
200 &HC8
201 &HC9
202 &HCA
203 &HCB
204 &HCC
205 &HCD
206 &HCE
207 &HCF
208 &HD0
209 &HD1
210 &HD2
211 &HD3
212 &HD4
213 &HD5
214 &HD6
215 &HD7
216 &HD8
217 &HD9
218 &HDA
219 &HDB
220 &HDC
221 &HDD
222 &HDE
223 &HDF
224 &HE0
225 &HE1
226 &HE2
227 &HE3
228 &HE4
229 &HE5
230 &HE6
231 &HE7
232 &HE8
233 &HE9
234 &HEA
235 &HEB
236 &HEC
237 &HED
238 &HEE
239 &HEF
240 &HF0
241 &HF1
242 &HF2
243 &HF3
244 &HF4
245 &HF5
246 &HF6
247 &HF7
248 &HF8
249 &HF9
250 &HFA
251 &HFB
252 &HFC
253 &HFD
254 &HFE
255 &HFF

K

NOMBRES RESERVADOS

Las palabras siguientes tienen un significado reservado. Aquellas marcadas con un asterisco indican ampliaciones futuras.

ABS	DEFSTR	KEY	PAINT	STRING$
AND	DELETE	KILL	PDL	SWAP
ASC	DIM	LEFT$	PEEK	TAB(
*ATTR$	DRAW	LEN	PLAY	TAN
ATN	DSKF	LET	POINT	THEN
AUTO	DSKI$	LFILES	POKE	TIME
BASE	DSKO$	LINE	POS	TO
BEEP	ELSE	LIST	PRESET	TROFF
BIN$	END	LLIST	PRINT	TRON
BLOAD	EOF	LOAD	PSET	USING
BSAVE	EQV	LOC	PUT	USR
CALL	ERASE	LOCATE	READ	VAL
CDBL	ERL	LOF	REM	VARPTR
CHR$	ERR	LOG	RENUM	VDP
CINT	ERROR	LPOS	RESTORE	VPEEK
CIRCLE	EXP	LPRINT	RESUME	VPOKE
CLEAR	FIELD	LSET	RETURN	WAIT
CLOAD	FILES	*MAX	RIGHT$	WIDTH
CLOSE	FIX	MERGE	RND	XOR
CLS	FN	MID$	RSET	
*CMD	FOR	MKD$	RUN	
COLOR	*FPOS	MKI$	SAVE	
CONT	FRE	MKS$	SCREEN	
COPY	GET	MOD	SET	
COS	GO TO	MOTOR	SGN	
CSAVE	GOSUB	NAME	SIN	
CSNG	GOTO	NEW	SOUND	
CSRLIN	HEX$	NEXT	SPACE$	
CVD	IF	NOT	SPC(	
CVI	IMP	OCT$	SPRITE	
CVS	INKEY$	OFF	SQR	
DATA	INP	ON	STEP	
DEF	INPUT	OPEN	STICK	
DEFDBL	INSTR	OR	STOP	
DEFINT	INT	OUT	STR$	
DEFSNG	*IPL	PAD	STRIG	

RESUMEN DE LAS INSTRUCCIONES MSX-BASIC

En este cuadro se tratan todas las instrucciones, comandos y funciones BASIC someramente dándose la sintaxis (es decir la manera completa de escribir) de cada una de las instrucciónes y los comandos. Se utilizan dos símbolos, con el siguiente significado:

[...] todo lo que está entre corchetes es optativo, es decir: puede, eventualmente, prescindirse de ello;

<...> los datos encerrados entre estos signos deben ser rellenados por el mismo usuario.

En algunas expresiones se encuentran tres puntos como, por ejemplo, en:

CALL<nombre>[<literal>,<literal>...]

Los tres puntos que se encuentran al final de esta expresión quieren decir 'la expresión puede proseguirse de este modo' en cuyo caso podrán ser acogidos más litérales, separadas por comas.

Este cuadro da además la sintaxis completa de todas las funciones BASIC.

Aquí se adoptarán dos clases de notación, con el siguiente significado:

<X> variable numérica o expresión a rellenar por el usuario;

<X$> variable literal o expresión a rellenar por el usuario.

En cada orden se menciona la categoria: Instruccion, comando, variable de sistema, o funcion.

Despues de ejecutar un comando, el MSX-BASIC proyecta el 'prompt'. OK en la pantalla y espera hasta que se introduzca el siguiente comando o instruccion.

Una funcion o variable del sistema solo puede ser utilizada conjuntamente con un comando.

ABS(<X>)
Da el valor absoluto de <X>.
Categoría: función
Ejemplo: `PRINT ABS(-3)`

ASC(<X$>)
Da el código ASCII del primer signo de <X$>.
Categoría: función
Ejemplo: `PRINT ASC("MSX")`

ATN(<X>)
Da el arcotangente de<X>en radianes. El resultado se encontrará entre $-\pi/2$ y $\pi/2$.
Categoría: función
Ejemplo: `PRINT ATN(3)`

AUTO [<números de línea>] [,[<tamaño del paso>]
Genera automáticamente números de línea durante la carga de un programa. Si no se indica ningún <número de linea> se comenzará por el número 10 y, en caso contrario, por el <número de línea> indicado. Si no se indica <el tamaño del paso>, entonces se aumenta en diez sucesivamente el número de línea, en caso contrario, se aumenta, en el número de tamaño del <paso> indicado.
Si se genera automáticamente un número de línea que ya se utilizó antes, aparecerá un asterisco * de aviso.
El comando se termina con CRTL/C, es decir, manteniendo la tecla CTRL apretada mientras se pulsa C.
Categoría: comando
Ejemplo: `AUTO 100,50`

BASE(<X>)
Contiene la ubicación del llamado Video Display Processor (tablas VDP).
El cuadro siguiente da un resumen de las abreviaturas usadas.

tm	= modo texto		
gm	= modo gráfico	spat	= tabla de atributos de sprites
pt	= tabla de patrones	ct	= tabla de colores
spt	= tabla de patrones de sprites	nt	= tabla de nombres

(X)	*significado*			*(X)*	*significado*		
0	nt	en	tm1	11	ct	en	gm1
2	pt	en	tm1	12	pt	en	gm1
5	nt	en	tm2	13	spat	en	gm1
6	ct	en	tm2	14	spt	en	gm1
7	pt	en	tm2	15	nt	en	gm2
8	spat	en	tm2	16	pt	en	gm2
9	spt	en	tm2	17	spat	en	gm2
10	nt	en	gm1	19	spt	en	gm2

Categoría: variable del sistema
Ejemplo:
```
10 SCREEN 0
20 PRINT BASE(2):END
```

BEEP
Genera una señal acústica de corta duración.
El mismo efecto se logra con PRINT CHR$(7).
Categoría: instrucción
Ejemplo:
```
10 FOR K=1 TO 1000:PRINT K:NEXT K
20 BEEP:END
```

BIN$(<X>)
Convierte el número dado a la notación binaria.
<x> debe estar entre –32768 y 65535. Si <x> es negativo, obtenemos la llamada forma de 'el complemento a dos'.
Categoría: función
Ejemplo: `PRINT BIN$(100)`

BLOAD "<dev>: [<nombre de fichero>]" [,R] [,<desplazamiento>]
Sirve para cargar un programa escrito en código máquina que ha sido almacenado por medio del comando BSAVE.
Si se pone en lugar de <dev> el término CAS, es que se refiere a una lecto-grabadora de datos.
Se pueden usar las letras A hasta F en lugar de <dev>. Estas tienen entonces relación con las unidades 1 hasta 6.
El término <nombre de fichero> se refiere al nombre que se le había dado (6 signos como máximo, para la lecto-grabadora y 8 para la unidad de discos.
Si se indica la letra R, el programa será ejecutado inmediatamente después de que esté cargado.
Si se indica un número (entero) de aviso, entonces se comenzará por la dirección de salida que sea igual a la antigua + el número de aviso.
Categoría: comando
Ejemplo: `BLOAD "CAS:TEST",R,&H20`

BLOAD "<dev>: [<nombre de fichero>]", S
Ocurre lo mismo que en el comando anterior pero, en este caso, la S se refiere a la memoria RAM de vídeo. Tenga en cuenta que <dev> aquí sólo puede ser un disquete.
<dev> solo puede ser A hasta F.
Categoría: comando
Ejemplo: `BLOAD "A:TEST",S`

BSAVE"<dev>: [<nombre de fichero>]", <dirección inicial, dirección final> [,<dirección inicial para la puesta en marcha del programa>]
Sirve para almacenar un programa escrito en código máquina, en un soporte externo. Los términos, <dev> y <nombre de fichero>, ya han sido aclarados en BLOAD. Las direcciones inicial y final indican el espacio de memoria que será copiado en el soporte externo.
Categoría: comando
Ejemplo: `BSAVE "CAS:TEST",&HC000,&HE0FF,&HC020`

BSAVE"<dev>: [<nombre de fichero>]", <dirección inicial>, <dirección final>, S
Como el anterior, pero, ahora se trata de la memoria RAM de vídeo.
<dev> puede ser aquí, sólo, un disquete.
Categoría: comando
Ejemplo: `BSAVE "A:TEST",&H0,&HFFFF,S`

CALL<nombre> [(<literal>, <literal>...)]
Comando general para ejecutar ciertas subrutinas que se encuentran almacenadas en un cartucho ROM.
Las cadenas indicadas son constantes alfanuméricas, mediante las cuales podemos pasar parámetros. En lugar de la palabra CALL podemos usar el signo (_) 'underscore'.
Categoría: instrucción
Ejemplo: `_TALK`

CALL COM([<X>:],GOSUB<Y>)
Esta instrucción la podemos usar si la interface RS232C está presente. <X> indica el número de la interface. El valor por defecto (inglés: default value) es 0. <Y> indica un número de línea. Con esta instrucción se indica que hay que saltar a la subrutina que empieza con el número <y> en el momento en que se haga una llamada a la interface. Esta debe ser activada con anticipación por medio de la instrucción CALL COMON.
Categoría: instrucción
Ejemplo:

```
 10 OPEN "COM:" FOR INPUT AS#1
 20 CALL COMON ("")
 30 CALL COM(,GOSUB 60)
 40 PRINT "PROGRAMA PARA RECIBIR DATOS"
 50 GOTO 50
 60 PRINT "AHORA RECIBO: ";
 70 A$=INPUT$(1,#1)
 80 PRINT A$
 90 RETURN
100 END
```

CALL COMBREAK (["<n>:"], <código de especificación>)
Sirve para trasmitir los llamados carácteres de interrupción. El código de especificación se usa aquí como signo para cortar la comunicación (número entre 3-32767). Este comando sólo está activado si la RS232C ya está abierta.
Con <n> se indica la interface correspondiente. El valor por defecto es 0.
Categoría: instrucción
Ejemplo:

```
10 OPEN "COM:" FOR OUTPUT AS#1
20 CALL COMBREAK(,3)
30 CLOSE
40 END
```

CALL COMDTR (["<n>:"],<0 o no-0>)
Con éste indicamos que la interface RS232C se desconecta temporalmente, eso, por lo menos, si la expresión <0 o no-0> es igual a 'no-cero'. Este comando sólo es válido cuando la RS232C ya está abierta.
Con <n> se indica la interface correspondiente. El valor de defecto es 0.
Categoría: instrucción
Ejemplo:

```
10 OPEN "COM:" FOR OUTPUT AS#1
20 CALL COMDTR(,0)
30 CLOSE
40 END
```

CALL COMINI [([<expresión literal>][,[<Rx velocidad en baudios>] [,[<Tx velocidad en baudios>][,[<límite de tiempo>]]]])]

Instrucción para inicializar la interface RS232C.
Significado de los campos:

<expresión literal>: véase abajo

<Rx velocidad en baudios>: velocidad (bit/segundo) con que recibe signos la interface. Si no se indica: 1200 baudios

<Tx velocidad en baudios>: velocidad (bit/segundo) con que envía signos la interface. Si no se indica: 1200 baudios.

<límite de tiempo>:tiempo (en segundos) que esperará el ordenador al acuse de conexión antes de dar la señal de error. El tiempo límite=0 significa que el ordenador esperará indefinidamente.
Si no se indica: 0
<expresión literal>: ésta tiene la forma:
"[<knl>:][<bl> [<pa>[<sl> [<x>[<hs>[<ilf>[<slf>[<ss>]]]]]]]]"
significado:

<knl> número de canal: número de la interface RS232C. Sólo es necesario si se tienen más de una.
Defecto: 0

<bl> longitud de byte: longitud (en bits) de la unidad de datos enviada de una vez (byte). A rellenar "5", "7" o "8".
Defecto: "8".

<pa> paridad
"E" = paridad par (Even)
"O" = paridad impar (Odd)
"I" = no interesa (Ignore)
"N" = no hay paridad (No)
Defecto: "E"

<sl> longitud de pausa: longitud (en bits) de la pausa que hay entre el envío de dos bytes
"1" = 1 bit
"2" = 1.5 bit
"3" = 2 bits
Defecto: "1"

<x> Dirección XON/XOFF
"X" = dirigir
"N" = no dirigir
Defecto: "X"

<hs> acuse de conexión CTS-RTS
"H" = acusada
"N" = no acusada
Defecto: "H"

<ilf> intercalar signo de alimentación de línea, cada vez que se recibe un signo de retorno de carro
"A" = intercalar

"[0 :] [8 [n [1 [X [H [N [N [N]]]]]]]]]"

Canal n°. Si sólo un canal puede omitirse esta indicación. El llamado 'default' es 0

Longitud de datos
"5" – 5 bits
"6" – 6 bits
"7" – 7 bits
"8" – 8 bits

Tabla de paridades
"E" – par
"O" – impar
"I" – se ignora
"N" – no hay

Bits de parada (longitud)
"1" – 1 bit
"2" – 1.5 bit
"3" – 2 bits

Acuse de conexión CTS-RTS
"H" – hay acuse
"N" – no hay acuse

Insertar alimentación de línea LF
Si se recibe CR
"A" – insertar
"N" – no insertar

Insertar alimentación de línea LF
Si se envía CR
"A" – insertar
"N" – no insertar

Control Shift si Shift no
(no está permitido si)
longitud de datos=7 bits
"S" – activa el control
"N" – no

"N" = no intercalar
Defecto: "N"

<slf> envío de signo de avance de línea después de cada retorno de carro
"A" = enviar
"N" = no enviar
Defecto: "N"

<ss> control con/sin shift
"S" = controlar
"N" = no controlar
Defecto: "N"

Categória: instrucción
Ejemplo: `CALL COMINI("0:7N2XHAAN",75,1200)`

CALL COMON("[<n>:]")
CALL COMOFF("[<n>:]")
CALL COMSTOP("[<n>:]")

Con <n> se indica la correspondiente interface.
Por medio de cierto número de comandos CALL puede dejar que un programa BASIC ejecute una subrutina, automáticamente, tan pronto como se reciba un signo a través de la interface RS232C. El funcionamiento de este comando es totalmente análogo al de ON <suceso> GOSUB número de línea.
Categoría: instrucción
Ejemplo: véase CALL COM

CALL COMSTAT(["<n>:"], <número de variable>)

Con <n> se indica la correspondiente interface RS232C.
Si durante la comunicación por la interface RS232C se presenta un Device I/O error, la variable adquiere un valor que indica la falta ocurrida.
La siguiente tabla ilustra las posibilidades existentes, en cierto número de casos se han adoptado los términos ingleses estándar.

Bit n	*significado*
15	Buffer overflow error (memoria intermedia llena)
	0-no
	1-sí
14	Time out error (falta tiempo)
	0-no
	1-sí
13	Framing error (error de estructuración)
	0-no
	1-sí
12	Over run error
	0-no
	1-sí
11	Parity error
	0-no
	1-sí
10	Tecla de control de interrupción pulsada
	0-no
	1-sí
9	no usado
8	no usado
7	Clear to Send
	0-no
	1-sí
6	Timer/counter output-2
	0-contador de tiempo parado
	1-contador asignado
5	no usado
4	no usado
3	Data Set Ready
	0-no
	1-sí

Bit n	*significado*
2	Break detect 0-no 1-sí
1	Ring Indicator 0-no 1-sí
0	Carrier Detect 0-no 1-sí

Categoría: instrucción

Ejemplo:
```
10 OPEN "COM:" FOR INPUT AS#1
20 CALL COMSTAT(,A%)
30 A$="0000000000000000"+BIN$(A%)
40 PRINT RIGHT$ (A$,16)
50 CLOSE
60 END
```

CALL COMTERM [("<n>:")]

Empieza el llamado 'modo simulador de terminal' (terminal emulator mode) de la interface RS232C. Con <n> se indica la correspondiente interface RS232C; el valor de defecto es 0.

El canal RS232C debe ser cerrado primeramente mediante la instruccion CLOSE, antes de ejecutar CALL COMTERM.

Las teclas F6, F7 y F8 tienen un significado especial:

F6 conecta/desconecta el llamado 'modo literal'. En este estado se imprimen los caracteres 0 hasta 31 con el signo ∧ seguido de la letra que se adquiere al sumarle 64 a este código.

F7 modo half/full duplex. En el 'modo half duplex', cada signo que se introduce, es también enviado por el canal RS232C.

F8 conecta/desconecta el llamado 'eco-impresor'.

Categoría: comando

Ejemplo: `CALL COMTERM`

CALL FORMAT

Para formatear un disquete no usado aún.

Después del comando CALL FORMAT se pregunta en qué unidad se encuentra el disco a formatear.

¡Cuidado! un disco ya utilizado será borrado por completo por el comando FORMAT.

Categoría: comando

Ejemplo: `CALL FORMAT`

CALL SYSTEM

Orden para salir del MSX-BASIC y darle el comando al sistema MSX-DOS. Se supone entonces que el sistema MSX-DOS ya ha sido puesto en marcha.

Categoría: comando

Ejemplo: `CALL SYSTEM`

CDBL(<X>)

Convierte<X>en un número de precisión doble.

Categoría: función
Ejemplo: `PRINT CDBL(7/6)`

CHR$(<código ASCII>)
Da el signo de escritura que corresponde al<código ASCII>.
Categoría: función
Ejemplo: `PRINT CHR$(65)`

CINT(<X>)
Redondea al valor de<X>en un número entero. El valor debe encontrarse entre –32768 y 32767.
Categoría: función
Ejemplo: `PRINT CINT(7/6)`

CIRCLE [STEP] (<x,y>), <r> [,[<color>] [, [<ángulo inicial>] [,[<ángulo final>] [,<combadura>]]]]
Genera una elipse o, (de excentricidad 1.4) un círculo. El punto central se indica con <x, y>, <r> indica el radio. Puede dibujarse, eventualmente, sólo una parte, indicando un ángulo inicial y uno final (en radianes).
Si se omite STEP, será tomado automáticamente el sistema normal de coordenadas. STEP lo que hace es correr el sistema de coordenadas hasta el punto citado después de STEP.
<achatamiento> es un número que indica la relacion entre el radio horizontal y el radio vertical.
Categoría: instrucción
Ejemplo:

```
10 SCREEN 2
20 CIRCLE (127,95),50,,,1.4
30 GOTO 30
```

CLEAR [<espacio de memoria para literales> [,<dirección más alta>]]
Con éste se hacen todas las variables igual a cero y, además, se reserva un espacio de memoria para el almacenamiento de caracteres (litérales, letras).
La 'dirección más alta' determina el puesto más alto para usar en BASIC. Finalmente, CLEAR tiene como consecuencia que todos los ficheros que estén abiertos, se cierren. Observe que sin CLEAR el ordenador reserva solamente 200 bytes para el almacenamiento de letras etc.
Hacemos notar ademas, que las funciones programadas anteriormente, al igual que las variables estandarizadas que habían sido indicadas con DEF FN también son borradas.
También se borran todas las tablas que habían sido indicadas por medio de una instrucción DIM.
Categoría: instrucción
Ejemplo:

```
10 A=10:B$="TEST"
20 PRINT A,B$
30 CLEAR
40 PRINT A,B$:END
```

CLOAD ["<nombre de fichero>"]
Comando para introducir un programa, eventualmente con nombre (6 letras y/o cifras como máximo), desde un cassette. Así CLOAD 'PROG4' produce la carga, desde un cassette, de un programa que lleve por nombre PROG4. Si algo marcha mal hay que interrumpir la carga con CTRL/STOP, rebobinar hacia atrás la cinta y volver a empezar.

Si se omite el nombre del fichero, será cargado el primer programa que se encuentre en la cassette.
Categoría: comando
Ejemplo: `CLOAD "TEST"`

CLOAD? ["<nombre de fichero>"]
Comando para comparar un programa en la cassette con el programa en la memoria. Si todo está bien, aparecerá 'Ok'. Si se ha cometido alguna falta, aparecerá 'Verify error'.
Categoría: comando
Ejemplo: `CLOAD?"TEST"`

CLOSE [#] [<número de fichero>] [,[#]<número de fichero>...]
Cierra los ficheros que lleven los números indicados. Si no se indica ningún número, serán cerrados todos los ficheros.
Categoría: instrucción
Ejemplo:

```
10 MAXFILES=1
20 OPEN "CAS:TEST" FOR OUTPUT AS#1
30 A$="MSX"
40 PRINT#1,A$
50 CLOSE#1
60 END
```

CLS
Limpia la pantalla.
Categoría: instrucción
Ejemplo:

```
10 CLS
20 END
```

COLOR[<color de superficie>][,<color de fondo>][,<bordes>]
Con esta instrucción se puede determinar el color de la parte indicada. El cuadro siguiente nos enseña los colores estándar.

	Color		*Color*
0	transparente	8	rojo
1	negro	9	rojo claro
2	verde	10	amarillo obscuro
3	verde claro	11	amarillo claro
4	azul obscuro	12	verde obscuro
5	azul claro	13	magenta (morado)
6	rojo obscuro	14	gris
7	cianógeno	15	blanco

CONT
Con este comando se prosigue la ejecución del programa a partir del lugar en que se había interrumpido con CTRL/STOP o con la instrucción STOP o la instrucción END.
Categoría: comando
Ejemplo: `CONT`

COPY "[<dev:>]<nombre de fichéro>" TO "[<dev>:]<nombre de fichéro>"
Con esto podemos copiar un fichéro.

<dev> solo tiene relación con una ductora A hasta F.
Categoría: instrucción
Ejemplo: `COPY "A:TEST.ASC" TO "B:"`

COS(<X>)
Da el coseno de <X>. <X> en radianes.
Categoría: función
Ejemplo: `PRINT COS(3.1415/6)`

CSAVE"<nombre de fichero>"[,<velocidad en baudios>]
Comando para almacenar un programa en una cassette (véase también CLOAD).
Las velocidades en baudios son:

1 1200 baudios
2 2400 baudios

Si se omite esta especificación, la velocidad que se adopte será de 1200 baudios.
La velocidad en baudios puede fijarse también por medio de SCREEN.
Categoría: comando
Ejemplo: `CSAVE "TEST"`

CSNG(<X>)
Convierte el valor de<X>en número de simple precisión.
Categoría: función
Ejemplo: `PRINT CSNG(9/7)`

CSRLIN
Da la coordenada y de la posición en que se encuenta el cursor.
Categoría: función
Ejemplo:

```
10 SCREEN 0:LOCATE 10,20:PRINT CSRLIN
20 END
```

CVI(<literal de 2 bytes>)
CVS(<literal de 4 bytes>)
CVD(<literal de 8 bytes>)
Estas funciones decodifican los literales insertados respectivamente a los valores de entero (CVI), (CVS) o con precisión doble (CVD). Los literales deben estar codificados según las funciones correspondientes (MKI$, MKS$ y MKD$, véanse éstas) o ser introducidos a partir de un fichero directo. Podemos utilizar estas funciones para localizar números almacenados en un fichero directo. En los comandos FIELD sólo pueden indicarse variables literales; con estas funciones podemos hacer salir de un registro números codificados en forma de literales.
Categoría: función
Ejemplo: `PRINT CVD(D$);CVS(S$);CVI(I$)`

DATA<nl>[,<n2>...]
Con esta instrucción puede introducirse en un programa una lista de constantes fijas que podrán luego ser recuperadas una por una por medio de la instrucción READ.
La lista puede constar, tanto de valores numéricos, como de literales (entre comillas (")) y tienen que estar separados por comas. READ saca los valores en orden de secuencia. Puede volver a hacer salir desde el principio una lista de DATA, ejecutando primero la instrucción RESTORE.

Categoría: instrucción

Ejemplo:
```
10 READ A,B,C:PRINT A,B,C
20 DATA 5,6,7
30 END
```

DEF FN <nombre de función>[(lista de argumentos>)]=<definición de función>

Sirve para definir funciones propias. El nombre con el cual hay que realizar la llamada de la función es el <nombre de función> precedido de FN. En funciones de literales el nombre termina por $. Si se quiere pueden darse argumentos a la función, los cuales serán utilizados en la <definición de función>. Los nombres de los argumentos sólo sirven para definir la función y no tienen ningún influjo en las eventuales variables del programa principal que tengan el mismo nombre. Los argumentos tienen que ir separados entre sí por comas.

En la <definición de función> pueden incluirse también nombres de variables que no estén incluidos como argumentos de la <lista de argumentos>, pero que hubieran recibido un valor en el programa principal, ya antes de que se haya realizado la llamada de la función. La <definición de función> sólo puede ocupar una línea como máximo.

Categoría: instrucción

Ejemplo:
```
10 DEF FNAB(X,Y)=X^2+Y*Z
20 Z=5
30 I=2:J=3
40 T=FNAB(I,J)
50 PRINT T
60 END
```

DEF <tipo> <indicación de letra>

Esta instrucción sirve para que todas las variables que empiezan con la(s) <letra(s)> dada(s) pertenezcan al <tipo> indicado. Los tipos posibles son:

INT entero
SNG simple precision
DBL precisión doble (con dos cifras de aproximación)
STR literal

Los signos que especifican un tipo (o sea %,#,$) tienen prioridad sobre la instrucción DEF.

Ejemplos:

`10 DEFDBL A-E`	Todas las variables que comienzan por A, B, C, D o E son variables de precisión doble.
`10 DEFSTR A`	Todas las variables que empiezan con la letra A son variables de literales.

Categoría: instrucción

Ejemplo:
```
10 DEFINT I
20 I=3/2:PRINT I
30 END
```

DEF USR [<número>]=<dirección de memoria>

Sirve para especificar la dirección en que comienza una subrutina en lenguaje máquina. El <número> debe ser uno de los comprendidos entre 0 y 9 y es asignado como nombre a la rutina de lenguaje máguina. Si se omite el <número> será adoptado el O. A base de este <número> y con ayuda de la función USR puede realizarse la llamada de la subrutina al respecto en lenguaje máquina.

Categoría: instrucción

Ejemplo:
```
10 DEF USR0=1000
20 X=USR0(9*2)
30 END
```

DELETE [<número de línea>] [-<número de línea>]

Hace desaparecer todas las líneas indicadas. Después de la ejecución de este comando se vuelve siempre al modo de comandos de BASIC.

Categoría: comando

Ejemplo: `DELETE 10`

DIM <nombre de tabla> (<índice máximo>) [,<nombre de tabla>...]

Crea espacio de memoria para las tablas especificadas e inicializa los elementos de la tabla en 0. Cuando se hace referencia a una tabla que no haya sido creada de antemano por una instrucción DIM, se tomará 10 como índice máximo. El índice más bajo es siempre 0. Las tablas pueden borrarse con ERASE.

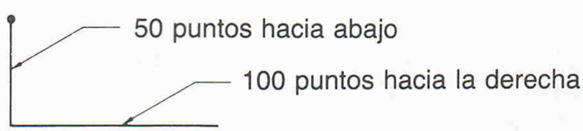
Categoría: instrucción

Ejemplo:
```
10 DIM A(20)
20 FOR K=1 TO 20:A(K)=K:NEXT K
30 FOR K=1 TO 20:PRINT A(K):NEXT K
40 END
```

DRAW <literal>

Por medio de DRAW puede indicarse toda clase de líneas rectas, con ayuda de códigos sencillos. Los códigos forman siempre un litéral.

Ejemplo:
```
10 SCREEN 2
20 DRAW "D50R100"
30 GOTO 30
```



Este indica dos líneas rectas unidas entre sí. La primera pasa por 50 puntos de imagen hacia abajo ('down 50' o abreviado D50) y la segunda por 100 puntos hacia la derecha (R100).

Para cada segmento podemos indicar un color, por medio de la letra C y un código de color, por ejemplo:

```
DRAW "C8D50C10R100"
```

En la tabla siguiente se da un cuadro de las posibilidades existentes:

Código	*Significado*
S	Indica la escala. La S va seguida de un número. La escala coincide con el número 14.
A	La A va seguida de 0, 1, 2 o 3. Con esto trastocamos el sistema de las coordenadas en pasos de 90°. Sin la indicación de A, el ordenador adoptará A0.
C	Indica el color. Véase el ejemplo de arriba.
M	Para dibujar líneas oblicuas. La M va seguida de dos números separados por una coma, por ejemplo: M30, 50 En este ejemplo será trazada una línea, desde la última posición hasta el punto que obtengamos al sumar 30 a la coordena de X y 50 a la coordenada Y. Los valores x e y pueden ser también negativos.

U U se refiere a 'up' que quiere decir 'hacia arriba'. Por ejemplo:

U30

indica que desde la última posición hay que trazar una línea hacia arriba, a través de 30 puntos.

D D se refiere a 'down'. Para el trazado de una línea hacia abajo.

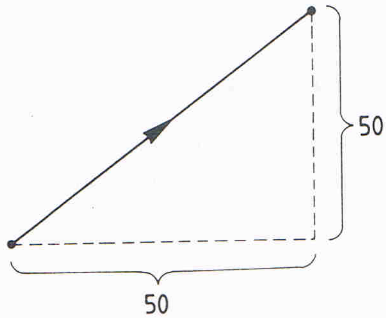
R R se refiere a 'right'. Para el trazado de una línea hacia la derecha.

L L se refiere a 'left'. Para el trazado de una línea hacia la izquierda.

E Para un línea trazada bajo un ángulo de 45° hacia el lado superior derecho. Por ejemplo:

E50

da como resultado:



F Para una línea trazada bajo un ángulo de 45° hacia el lado inferior derecho (véase también E).

G Para una línea trazada bajo un ángulo de 45° hacia el lado superior izquierdo (véase también E).

H Para una línea trazada bajo un ángulo de 45° hacia el lado inferior izquierdo (véase también E).

Obsérve que las líneas se trazan siempre a partir de la última posición alcanzada. Para obtener una posición inicial fija podemos servirnos de BM x,y en que x e y representan las coordenadas iniciales.

Ejemplo:

```
DRAW "BM50,50D30"
```

da una línea hacia abajo de 30 puntos a partir del punto (50,50).
También se puede adoptar el último punto alcanzado, como punto inicial de una nueva línea, poniendo delante del código una N. El literal puede indicarse también por medio de una variable literal.

Ejemplo: ....

```
50 B$="BM50,50D30"
60 DRAW B$
```

En una instrucción DRAW también puede reproducirse una parte del literal, por medio de una variable. En la parte de la instrucción DRAW colocaremos entonces una X.

Ejemplo: ....

```
50 B$="BM50,50D30"
60 DRAW "R100XB$"
```

Finalmente, las cantidades que aparecen en el literal DRAW también pueden indicarse por medio de una variable. La variable se colocará entre los signos (=) y (;).

Ejemplo: ...

```
50 K=30
60 DRAW "B10,10R=K;"
```

Categoría: instrucción
Ejemplo:

```
10 SCREEN 2
20 DRAW "BM80,80H20"
30 GOTO 30
```

DSKF(<número de unidad de disco>)
Esta función da la cantidad de espacio que queda en un disco.
Los unidades ductoras de discos están numerados del modo siguiente:

0 = unidad inexistente
1 = unidad A
2 = unidad B
. .
. .
. .
6 = unidad F

Categoría: función
Ejemplo:

```
10 PRINT DSKF(1)
20 END
```

DSKI$(<número de unidad de discos>,<número de sector>)
Esta función carga el sector indicado en la sección de la memoria que se establece por medio de los lugares &HOF351 y &HOF352.
Los números de 0 hasta 6 indican las unidades A hasta F.
Categoría: función
Ejemplo:

```
 10 SCREEN 0:WIDTH 40:COLOR 15,4,4:DEFINT I
 20 PRINT "COPIA DE DISCO, POR SECTOR"
 30 PRINT:PRINT
 40 PRINT "INTRODUZCA EL ORIGINAL EN LA UNIDAD A:"
 50 PRINT
 60 PRINT "DISCO NUEVO EN LA UNIDAD B:"
 70 PRINT
 80 PRINT "PULSE CUALQUIER TECLA."
 90 A$=INPUT$(1)
100 PRINT:PRINT
110 PRINT "TRACK :"
120 PRINT
130 PRINT "SECTOR:"
140 FOR I=1 TO 719
150 A$=DSKI$(1,I)
160 LOCATE 8,10:PRINT INT(I/9)
170 LOCATE 8,12:PRINT IMOD9
180 DSKO$ 2,I
190 NEXT I
200 END
```

DSKO$<número de unidad>,<número del sector>
Traslada el contenido de la memoria (indicado por medio del contenido de las direcciones de memoria &HOF351 y &HOF352) al sector especificado.
Los números de 0 hasta 6 indican las unidades A hasta F.
Categoría: función
Ejemplo: vease DSKI$

END

Finaliza un programa. Esta instrucción puede aparecer en cualquier parte del programa, al final, en cambio, es optativa, es decir, no es obligatoria, pues todos los archivos que estén todavía abiertos se cerrarán.

Categoría: instrucción

Ejemplo:
```
10 INPUT A
20 IF A<5 THEN END
30 END
```

EOF (<número de fichero>)

Determina si ha sido alcanzado el final de un fichero (file), durante la entrada de datos.

Categoría: función

Ejemplo:
```
IF EOF(2) THEN CLOSE#2
```

ERASE <nombre de tabla> [,<nombre de tabla>...]

Con ERASE podemos borrar una tabla, de modo que nos quede algún espacio de memoria libre (véase también DIM).

Categoría: instrucción

Ejemplo:
```
 10 DIM K(100),X$(50)
    ...
500 ERASE K,X$
```

ERR y ERL

Si en un programa se comete un error, la variable ERR da el número de dicho error (véase apéndice G, Cuadro de mensajes de error) y ERL contendrá el número de linea en el que se ha cometido la falta. ERR y ERL pueden ser utilizadas en una rutina de corrección de errores escrita por el usuario mismo (véase también ON ERROR GOTO), generalmente en construcciones IF...THEN.

Categoría: funcíon

Ejemplo:
```
10 ON ERROR GOTO 50
20 A=25:PRINT A
30 B=A/0
40 END
50 PRINT ERL
60 RESUME NEXT
```

ERROR <número de error>

Visualiza en la pantalla el código de error correspondiente al <número de error> indicado. El <número de error> tiene que ser mayor que 0 y menor que 255. Ejemplo: teclee ERROR 11. En la pantalla aparecerá /0Error.

No todos los números de error comprendidos entre 0 y 255 son utilizados por BASIC. Hay números libres que pueden ser utilizados para generar mensajes de error propios.

Del siguiente ejemplo puede deducirse que no siempre se imprime el mensaje de error estándar, sino que nosotros podemos también hacer imprimir un mensaje de error redactado por nosotros mismos.

Categoría: funcíon

Ejemplo:
```
10 ON ERROR GOTO 400
20 INPUT "X":-",A
30 IF A>100 THEN ERROR 210
   ...
```

```
...
400 IF ERR=210 THEN PRINT "¡100 COMO MAXIMO!"
410 RESUME 20
```

EXP(<X>)
Calcula la potencia e de (<X>)
Categoría: función
Ejemplo: `PRINT EXP(1)`

FIELD[#]<número de fichero>,<anchura de campo>AS<variable literal>
Este nos da acceso a la memoria intermedia de un fichero directo.
En un programa BASIC podemos leer o describir la memoria intermedia sirviéndonos de las variables literales nombradas en la parte AS.
Categoría: instrucción
Ejemplo: `FIELD #1,20 AS A$,10 AS B$`

Con esto se asignan las 20 primeras posiciones de la memoria intermedia del fichero 1 a A$ y las 10 posiciones siguientes a B$. Si ejecutamos ahora un comando GET podemos hacer salir el registro introducido en A$ y B$. Podemos introducir un nuevo registro dando un valor nuevo a A$ y B$, en un comando LSET o RSET, y ejecutando, a continuacin, un comando INPUT. FIELD no lee ni escribe él mismo en el disco.
La suma de las anchuras de campo de un comando FIELD no puede ser mayor que la longitud del registro del fichero (indicada en el comando OPEN).

FILES ["<nombre de fichero>"]
Da un cuadro del ficheros que se encuentran en el disco.
Categoría: comando
Ejemplo: `FILES`

FIX(<X>)
Da la parte entera de (<X>).
Categoría: función
Ejemplo: `PRINT FIX(1.7)`

FOR...NEXT
La frase completa reza:

FOR<variable>=<n>TO<m>[STEP<k>]
instrucción 1
instrucción 2
...
...
NEXT[<variable>] [,<variable>...]

en que n, m y k son expresiones numéricas.
Todas las instrucciones comprendidas entre FOR y NEXT son ejecutadas repetidamente, tantas veces como sea necesario para que el numerador (=<variable>) sobrepase el valor de m. El valor inicial del numerador es n y a éste se le adiciona k cada vez que hayan sido ejecutadas todas las instrucciones. Si k no está especificada el numerador es aumentado en 1 cada vez.

Las instrucciones FOR...NEXT pueden, a su vez, contener otras instrucciones FOR...NEXT.
Categoría: instrucción
Ejemplo:
```
10 FOR K=1 TO 10:PRINT K:NEXT K
20 END
```

FRE(0) o FRE(<" ">)
Si el argumento es 0 entonces FRE dará la cantidad de espacio de memoria no utilizado en bytes. Si el argumento es una variable literal entonces FRE dará la cantidad de bytes libres que queda en el espacio de memoria reservado a las variables literales. Este último espacio puede ser adaptado a base de CLEAR.
Categoría: función
Ejemplo:
```
PRINT FRE(0)
```

GET [#]<número de fichero>[,<número de registro>]
Por medio de éste podemos introducir un registro de un fichero directo en la memoria intermedia de dicho fichero. A continuación podremos utilizar el contenido del registro en un programma, si se ha dado un comando FIELD, antes del fichero, por medio de las variables literales que se indican en el comando FIELD.
Cada registro de un fichero directo tiene un número. Después de ser introducido un registro será almacenado internamente el número del mismo y en el comando GET que venga a continuación será introducido automáticamente el registro siguiente. Sin embargo, si en el comando GET se ha dado un número de registro, el registro será introducido con ese número.
El número de registro del último registro introducido puede localizarse por medio de la función LOC; véase ésta.
Categoría: instrucción
Ejemplo:
```
10 OPEN "VBD.DAT" AS#1
20 FIELD #1,2 AS A$,10 AS B$
30 FOR K%=1 TO 10
40 GET #1,K%
50 PRINT CVI(A$);B$
60 NEXT
70 CLOSE #1
80 END
```

GOSUB<número de línea>RETURN[<número de línea>]
Esta instrucción hace que sea posible saltar a una subrutina. El <número de linea> en el número de la primera línea de la subrutina. En la subrutina debe aparecer alguna vez RETURN ya que ésta es la encargada de hacer que se regrese a la instrucción que sigue a la instrucción GOSUB. Las subrutinas, a su vez, pueden referirse a otras subrutinas.
Categoría: instrucción
Ejemplo:
```
10 GOSUB 40
20 GOSUB 40
30 END
40 PRINT "SUBROUTINE"
50 RETURN
```

GOTO<número de línea>
La ejecución del programa prosigue desde el indicado <número de línea>.
Categoría: instrucción
Ejemplo:
```
10 GOTO 30
20 PRINT "A"
30 PRINT "13"
40 END
```

HEX$(<X>)
Da un literal que representa el valor hexadecimal de<X>.<X> es redondeado primero en un número entero.
<X>debe encontrarse entre -32768 y 65535.
Categoría: instrucción
Ejemplo: `PRINT HEX$(63)`

IF...THEN
La frase completa es:
IF<expresión lógica>[<operador lógico><expresión lógico>...]THEN <número de linea o instrucción(es)>[ELSE<número de linea o instrucción(es)>]
Nota: Con 'expresión lógica' se quiere decir 'condición lógica', sea cierta o no. En las <expresiones logicas> pueden aparecer como signos de relación (operador):

<	: menor que	< =	: menor que o igual a
>	: mayor que	> =	: mayor que o igual a
=	: igual a	< >	: distinto de

Si se colocan más instrucciones después de THEN o ELSE, éstas tienen que ir separadas por dos puntos (:).
La expresión total debe ocupar una línea.
En las expresiones puede utilizarsse OR, AND, NOT, XQR, EQV O IMP (véase el apéndice F).
Si toda la expresión comprendida entre IF y THEN es cierta, serán ejecutadas las instrucciones que siguen a THEN. Si después de THEN no hay instrucciones sino un número de línea, se continuará con la línea indicada.
Si la expresión comprendida entre IF y THEN no es cierta, serán ejecutadas las instrucciones de detrás de ELSE o se efectuará un salto a la línea indicada. Si ELSE no existe, se proseguirá con la línea que venga inmediatamente detrás de la instrucción IF...THEN.

Las instrucciones que vengan detrás de THEN o ELSE pueden contener otra expresión IF...THEN, siempre que ésta quepa en una línea en su totalidad. O sea, la instrucción:

```
IF X>Y THEN PRINT "MAYOR" ELSE IF Y>X THEN PRINT "MENOR" ELSE PRINT "IGUAL"
```

está permitida. Cada ELSE que aparece en una instrucción así va acoplado al IF más cercano que todavía no pertenezca a otro ELSE.
Categoría: instrucción
Ejemplo:
```
10 INPUT A
20 IF A<3 THEN PRINT "A<3"
30 END
```

INKEY$

Esta función se usa en la forma<variable literal>=INKEY$. La función controla si se ha pulsado alguna tecla del teclado. Si es así, entonces se asignará el signo de escritura correspondiente a la<variable literal>. Si no es así, la<variable literal> contendrá un literal vacío.

Categoría: instrucción

Ejemplo:

```
10 PRINT "PULSE UNA H: ";
20 A$=INKEY$
30 IF A$<>"H" THEN 20
40 PRINT A$
50 END
```

INP(<X>)

Nos retorna un byte que se lee en la puerta de salida número X.

Categoría: función

Ejemplo:

```
10 A=INP(&HA8)
20 A$="00000000"+BIN$(A):PRINT RIGHT$(A$,8)
```

INPUT ["<texto>";]<variable 1>[,<variable 2>...]

Introduce valores que deben ser escritos por el ususario en el teclado. En la pantalla aparece siempre un signo de interrogación (?)

Si se da un <texto>, aparecerá éste en la pantalla delante del signo de interrogación.

Los valores escritos son asignados a las <variables>. Tiene que haber tantos valores, separados por comas, como variables se incluyen en la instrucción INPUT. También están permitidas las variables literales, pero no pueden ponerse comas en el literal escrito. Si se comete un error al escribir los valores aparecerá el aviso de error? Redo y volverá a ejecutarse la instrucción INPUT.

Categoría: instrucción

Ejemplo:

```
10 INPUT A:PRINT A:END
```

INPUT #<número de fichero>,<variable 1>[,<variable 2>...]

Esta instrucción es comparable a INPUT pero, aquí, se trata de los valores de un fichero. El <número de fichero> es el número que ha sido asignado al fichero en cuestión por medio de OPEN. Los valores del fichero deben estar ordenados en el mismo orden que las variables en la instrucción INPUT. Durante la operación de entrada del literal, será determinado el final del mismo por una coma, RETURN o nueva línea (line feed). Sin embargo, si el primer signo del literal es el de comillas (", se tomará como final del mismo un segundo signo de comillas.

Categoría: instrucción

Ejemplo:

```
10 OPEN "CAS:DATA" FOR INPUT AS#1
20 IF EOF(1) THEN 40
30 INPUT#1,X$:PRINT X$:GOTO 20
40 CLOSE#1:END
```

INPUT$(<X>)/INPUT$(<X>,[#]<Y>)

Esta función se usa en la forma<variable literal>=INPUT$(<X>). La función introduce un literal, desde el taclado, de <X> signos de escritura. En lugar del teclado, puede indicarse también el fichero <Y>. Se aceptan todos los signos excepto CTRL/C.

Categoría: función

Ejemplo:

```
10 A$=INPUT$(4):PRINT A$:END
```

INSTR([I,]<X$>,<Y$>)

Investiga si <Y$> está contenida en <X$> y da su posición en ésta. Si <Y$> no aprarece en <X$>, INSTR dará el valor 0. INSTR empieza a buscar desde el principio de <X$> a no ser que [I,] se indique otra posición de salida. Si <Y$> es un litéral vacío, INSTR dará el valor 1.

Categoría: función

Ejemplo: `10 A$="ABCDEFG":PRINT INSTR(A$,"BCD"):END`

INT(<X>)

Esta función da el mayor entero que sea menor o igual que <X>.

Categoría: función

Ejemplo: `PRINT INT (3.7)`

INTERVAL ON
INTERVAL OFF
INTERVAL STOP

Sirven para indicar si se conecta, se desconecta o se mantiene una interrupción indicada por el reloj incorporado. El intervalo debe ser indicado en una instrucción ON INTERVAL GOSUB.

Después de la instrucción INTERVAL ON, y cuando se haya acabado el intervalo, el ordenador saltará a la subrutina indicada por ON INTERVAL GOSUB.

Categoría: instrucción

Ejemplo:

```
10 ON INTERVAL=300 GOSUB 40
20 INTERVAL ON
30 GOTO 30
40 K=K+6:PRINT K;"SEC"
50 RETURN
```

KEY<n>,"<literal de comandos>"

Para la asignación de un <literal de comandos> a una tecla de función. <n> es el número de la tecla de función (de 1 a 10). El <literal de comandos> tiene que ir entre comillas ("), y puede constar como máximo de 15 signos. En un <literal de comandos> puede incluirse un RETURN, añadiendo +CHR$(13).

Categoría: instrucción

Ejemplo: `KEY 1, "RUN"+CHR$(13)`

Si aquí se usa la tecla de función 1 el programa será ejecutado inmediatamente porque el comando RUN está cerrado por un RETURN.

KEY LIST

Da un cuadro de los literals de comandos de las 10 teclas de función.

Categoría: instrucción

Ejemplo: `KEY LIST`

KEY ON
KEY OFF

determina si se presentará en la pantalla el significado de las teclas de función o no.

Categoría: instrucción

Ejemplo: `KEY OFF`

KEY (<n>)ON
KEY (<n>)OFF
KEY (<n>)STOP
Sirven para controlar si ha sido pulsada una de las teclas de función. Este función se conecta con ON y se desconecta con OFF.
Con STOP, no saltará inmediatamente el ordenador a la subrutina, según ON KEY GOSUB, sino que esperará a que sea dado KEY (n) ON>.
Categoría: instrucción
Ejemplo:
```
10 KEY(1) ON
20 ON KEY GOSUB 50
30 GOTO 10
40 END
50 PRINT "KEY1":KEY(1)OFF:RETURN
```

KILL"<nombre de fichero>"
Para borrar un fichero en un disco.
Categoría: instrucción
Ejemplo: `KILL "A:X1.DAT"`

LEFT(<X$>,<I>)
Da un litéral compuesto de los primeros signos <I> de <X$>. <I> debe encontrarse entre 0 y 255.
Categoría: función
Ejemplo: `PRINT LEFT$("MSX",2)`

LEN(<X$>)
Da el número de signos de que consta <X$>. También se cuentan los espacios.
Categoría: función
Ejemplo: `PRINT LEN("MSX")`

[LET]<variable>=<valor>
Esta instrucción asigna un valor a una variable. El término LET puede omitirse eventualmente.
Categoría: instrucción
Ejemplo:
```
10 LET A=14:PRINT A
20 LET A=A+3:PRINT A:END
```

LFILES ["<nombre de fichero>"]
Da un cuadro de todos los ficheros que se encuentran en el disco.
Con LFILES se imprime este cuadro con ayuda de la impresora.
Categoría: comando
Ejemplo: `LFILES`

LINE [[STEP](<X1>,<Y1>)]- [STEP](<X2>,<Y2>)[,<Z>][,B[F]]
Traza una línea entre (X1,Y1) y (X2,Y2).
Por medio de STEP podemos indicar que el sistema de coordenadas es desplazado al punto citado. Si se indica una diagonal y se termina esta instrucción con B será dibujado un rectángulo con la diagonal acordada. Si se termina con BF el rectángulo será coloreado.
En los MODOS SCREEN 2 y 3, la coordinada X varía de 0 hasta 255.
La coordinada Y, en los MODOS SCREEN 2 y 3 varía entre 0 y 191. En los MODOS SCREEN 2 y 3 Z debe tener un valor entre 0 y 15.

Categoria: instrucción
Ejemplo:

```
10 SCREEN 5
20 LINE (0,0)-(255,191)
30 GOTO 30
```

LINE INPUT ["<texto>";]<variable literal>
Introduce un literal escrito en el teclado. El literal puede constar de toda clase de signos de escritura y puede tener una longitud arbitraria hasta un máximo de 255 signos.
Categoria: instrucción
Ejemplo:

```
10 LINE INPUT A$:PRINT A$
```

LINE INPUT #<número de fichero>, <variable de literal>
Esta función es comparable a la de LINE INPUT, pero introduce el literal que procede de un fichero. El <número de fichero> es el número que se le ha asignado al fichero correspondiente por medio de OPEN. Los literales son introducidos en su longitud completa hasta llegar a un RETURN.
Categoría: instrucción
Ejemplo:

```
10 OPEN "CAS:DAT" FOR INPUT AS#1
20 IF EOF(1) THEN 50
30 LINE INPUT#1,A$:PRINT A$
40 GOTO 20
50 CLOSE#1:END
```

LIST [<número de línea> [-[<número de línea>]]]
Este comando reproduce las líneas especificadas, en la pantalla. Si no se indica número de linea será reproducido todo el programa.
Categoría: comando
Ejemplo:

```
LIST
```

LLIST [<número de línea>[-[<número de línea>]]]
Lo mismo que LIST, pero aquí las líneas se imprimen con la impresora.
Categoría: comando
Ejemplo:

```
LLIST
```

LOAD"<dev>:<nombre de programa>"[,R]
Para cargar un programa en la memoria del ordenador.
Podemos utilizar como <dev>:
CAS, A, B, C, D, E, F, COM<número>.
Si se utiliza el magnetofón, entonces el programa debió haberse almacenado en cassette utilizando el comando SAVE. Si se añade una ",R" al comando, entonces el programa será comenzado directamente despues de cargarse en la memoria.
Categoría: comando
Ejemplo:

```
LOAD "CAS:DEMO"
```

LOC(<número de fichero>)
El resultado de esta función depende del modo en que ha sido abierto el fichero. Si se trata de un fichero directo, la función da como resultado el número del último registro introducido: véase GET. En otras clases de ficheros, la función produce la cantidad de bytes tratados (introducidos).
Categoría: función

Ejemplo:

```
10 OPEN "CAS:DAT" FOR OUTPUT AS#1
20 INPUT A$:PRINT#1,A$:PRINT LOC(1)
30 IF A$<>"END" GOTO 20
40 CLOSE#1:END
```

LOCATE[<X>] [,[<Y>][, [<exhibir/ocultar cursor>]]]

Desplaza el cursor hasta el lugar indicado. El cursor lo podemos exhibir u ocultar, indicando un 1 o un 0 respectivamente.
Solo es válido en los MODOS de texto 1 y 2. El alcance de X e Y depende de si se proporciona o no, la instrucción WIDTH.
Categoría: instrucción
Ejemplo:

```
10 SCREEN 0:CLS:LOCATE 10,10:PRINT "*"
```

LOF(<número de fichero>)

Esta función da la longitud (en bytes) del fichero indicado.
El fichero que se indica, debe ser abierto de antemano.
Categoría: función
Ejemplo:

```
10 OPEN "A:DAT" FOR INPUT AS#1
20 PRINT LOF(1):CLOSE#1:END
```

LOG(<X>)

Da los logaritmos naturales de <X>. <X> tiene que ser mayor que 0.
Categoría: función
Ejemplo:

```
PRINT LOG(1.5)
```

LPOS(<X>)

X puede ser aquí un número cualquiera. LPOS da la posición en la memoria de impresión.
Categoría: función
Ejemplo:

```
10 LPRINT:PRINT LPOS(0)
20 LPRINT "MSX";:PRINT LPOS(0)
```

LPRINT[[USING<formato de impresión>;]<expresión>]

Igual que PRINT o PRINT USING, pero con directrices de impresión.
Categoría: instrucción
Ejemplo:

```
LPRINT "MSX"
```

LSET<variable literal>=<expresión literal>
RSET<variable literal>=<expresión literal>

Con este comando podemos introducir datos en la memoria intermedia de un fichero directo. LSET cuida de que la variable que se encuentra a la derecha sea rellenada con espacios, de modo que entre en el campo el literal de la izquierda. En RSET el literal será corrido totalmente hacia la izquierda.
Categoría: instrucción
Ejemplo:

```
10 MAXFILES=1
20 OPEN "A:TEST" AS#1
30 FIELD #1,2 AS N1$, 4 AS N2$, 8 AS N3$, 20 AS N4$
40 INPUT "A%";A%
50 INPUT "B!";B!
60 INPUT "C#";C#
70 INPUT "D$";D$
```

```
 80 RSET N1$=MKI$(A%):RSET N2$=MKS$ (B!):RSET N3$=MKD$(C#):
    LSET N4$=D$
 90 PUT #1,1
100 A%=0:B!=0:C#=0:D$=""
110 PRINT A%;B!;C#;D$
120 GET #1,1
130 A%=CVI(N1$);B!=CVS (N2$):C#=CVD(N3$):D$=N4$
140 PRINT A%;B!;C#;D$
150 CLOSE #1
160 END
```

MAXFILES =<número>
Da el número máximo de ficheros que pueden estar abiertos simultáneamente. <número> debe estar entre 1 y 15.
En un programa se pueden mantener al máximo 15 ficheros abiertos al mismo tiempo. Sólo puede haber 6 ficheros abiertos al mismo tiempo en un programa.
Categoriá: instrucción
Ejemplo:

```
10 MAXFILES=2
20 OPEN "CAS:DEMO" FOR INPUT AS#1
30 OPEN "LPT:" FOR OUTPUT AS#2
40 INPUT#1,A$
50 PRINT#2,A$
60 CLOSE
```

MERGE "<dev>:[<nombre de fichero>]"
Agrega un programa almacenado en una memoria externa a los programas que se encuentran en la memoria. El programa tenía que estar almacenado en formato ASCII.
Las líneas de ese programa van siendo colocadas siguiendo el número de orden ascendente. Si dos líneas tienen el mismo número de orden, será cambiada la línea que ya se encuentre en la memoria por la nueva. El programa obtenido así permanecerá en la memoria tal como está.
Categoría: comando
Ejemplo: `MERGE "CAS:PROG1"`

MID$(<X$>,<I>[<,J>])
Da un litéral que constituye una parte de <X$>. El literal comienza en la posición <I> y tiene un número <J> de signos. Si <J> no es indicada, serán tomados todos los signos a partir de <I>.
Categoría: función
Ejemplo: `PRINT MID$("BASIC",2,3)`

MID$(<literal>,<primer carácter>[,<número de caracteres>]=<literal 2>
Con ayuda de este comando podemos modificar el <literal 1>. Para ello indicaremos cuántos caracteres hay que cambiar en el literal 2 y cuáles son, colocando el literal 2 en lugar del trozo indicado del literal 1.
Categoría: instrucción
Ejemplo:

```
10 A$= "FEPLIE"
20 MID$(A$,3,4)="LIPE"
```

Como resultado será asignada a A$ la litéral 'FELIPE'.

MKI$(<expresión de enteros>)
MKS$(<expresión de números reales>)
MKD$(<expresión de números reales de doble precisión>)
Estas funciónes producen un literal que contiene la versión codificada del número introducido. El literal puede ser descodificado con ayuda de las funciónes invertidas CVI, CVS y CVD; véanse éstas.
Los literales que obtenemos al aplicar estas funciónes pueden ser escritos en un registro de un fichero directo.
Categoría: instrucción
Ejemplo:
```
10 RSET D$=MKD$(WD$)
20 RSET S$=MKS$(WS!)
30 RSET I$=MKI$(WI%)
```

MOTOR ON
MOTOR OFF
Sirven para dirigir a distancia una lectógrabadora de datos. MOTOR ON tiene el mismo efecto que si se pulsa el botón de 'play' de la lectógrabadora y MOTOR OFF lo vuelve a desconectar.
Las palabras ON y OFF se pueden olvidar eventualmente. Entonces se pasa de un estado al otro.
Categoría: instrucción
Ejemplo:
```
10 MOTOR ON
20 CLOAD "DEMO"
```

NAME<nombre antiguo>AS<nombre nuevo>
Para redenominar ficheros
Categoría: instrucción
Ejemplo: `NAME "VBD" AS "EXP1"`

NEW
Sirve para borrar un programa almacenado en la memoria.
Se borran todas las variables, y se cierran todos los ficheros.
Categoría: comando
Ejemplo: `NEW`

OCT$(<X>)
Da un litéral que representa el valor octal de <X>. Primero se redondea <X> en número entero.
Categoría: función
Ejemplo: `PRINT OCT$(636)`

ON ERROR GOTO <número de línea>
Como consecuencia de esta instrucción, si en el programa se produce un error, este error no será presentado en la pantalla, como suele ocurrir, sino que se proseguirá en la línea cuyo número se ha especificado. Así se tiene la posibilidad de escribir uno mismo una rutina para la corrección de errores. El <número de línea> indica la primera línea de la rutina correctora. En esta rutina pueden utilizarse las variables ERR y ERL.
Por medio de RESUME se regresa de la rutina correctora al programa principal. La instrucción puede revocarse por medio de ON ERROR GOTO 0. Es aconsejable ejecutar en una rutina correctora la instrucción ON ERROR GOTO 0 para la correción de errores imprevistos.

Categoría: instrucción

Ejemplo:
```
10 ON ERROR GOTO 50
20 INPUT "TEXTO";A$
30 IF LEN(A$)>5 THEN ERROR 250
40 END
50 IF ERR=250 THEN PRINT "TEXTO MUY LARGO":RESUME 20
60 ON ERROR GOTO 0
70 END
```

ON <expresión entera>GOTO<número de línea>[,<número de línea>...]

Hace posible la continuación de la ejecución de un programa en un número de línea determinado por el valor de la <expresión entera>. Si la <expresión entera> da, por ejemplo, el valor 3, el programa proseguirá en el tercer número de línea que se haya especificado después de GOTO.

El valor de la <expresión entera> no puede ser negativo, cero o mayor que 255.

Si este valor es 0 o mayor, que la cantidad de números de línea expresados, el ordenador continuará el programa por la siguiente instrucción que sea ejecutable.

Categoría: instrucción

Ejemplo:
```
10 INPUT N
20 ON N GOTO 30,40
30 PRINT "1":GOTO 50
40 PRINT "2":GOTO 50
50 END
```

ON<expresión entera> GOSUB <número de línea>[,<número de línea>...]

Funcióna de la misma manera que la instrucción ON...GOTO, pero aquí los números de linea indicados tienen que constituir la primera línea de una subrutina.

Categoría: instrucción

Ejemplo:
```
10 INPUT N
20 ON N GOSUB 40,50
30 GOTO 60
40 PRINT "1":RETURN
50 PRINT "2":RETURN
60 END
```

ON INTERVAL=<tiempo> GOSUB <número de línea>

Con ésta se indica que el programa deberá saltar a la subrutina indicada después del tiempo de espera fijado por INTERVAL.

Categoría: instrucción

Ejemplo: Vease INTERVAL ON/OFF/STOP

ON KEY GOSUB <número de línea> [,<número de línea>...]

Indica a qué subrutina hay que saltar si se pulsa una de las teclas comprendidas entre F1 y F10.

Categoría: instrucción

Ejemplo: Vease KEY(<n>) ON/OFF/STOP

ON SPRITE GOSUB <número de línea>[,<número de línea>...]
Indica a qué subrutina hay que saltar cuando dos sprites se cubren parcialmente uno al otro.
Categoría: instrucción
Ejemplo: Vease SPRITE ON/OFF/STOP

ON STOP GOSUB <número de línea>[,<número de línea>...]
Indica a qué subrutina hay que saltar cuando el programa es interrumpido por medio de CTRL/STOP.
Categoría: instrucción
Ejemplo: Vease STOP ON/OFF/STOP

ON STRIG GOSUB<número de línea>[,<número de línea>...]
Indica a qué subrutina hay que saltar. Para las líneas 1, 2, 3, 4 y 5 hay que tener en cuenta:

no.	*botón de fuego o barra espaciadora*
1	barra espaciadora
2	joystick 1, botón de fuego 1
3	joystick 2, botón de fuego 1
4	joystick 1, botón de fuego 2
5	joystick 2, botón de fuego 2

Categoría: instrucción
Ejemplo: Vease STRIG (<X>) ON/OFF/STOP

OPEN "<dev>:[<nombre de fichero>]"[FOR<tratamiento>] AS [#]<número>
Sirve para abrir un fichero (file).
En el lugar del <dev> puede ponerse lo siguiente:

CAS:	lectógrabadora de datos
CRT:	pantalla textual
GRP:	pantalla gráfica
LPT:	impresora
COM[<Z>]:	RS232 interface de comunicación.
A hasta F:	unidad de disco 1 hasta 6

En lugar de <nombre de fichero> puede tomarse un nombre cualquiera, compuesto de 8 caracteres como máximo más una extension de tres caracteres. (con excepcion de la lecto-grabadora, con el cual solo se pueden tomar 6 caracteres, sin incluir la extensión).
Si en el lugar de <tratamiento> se pone OUTPUT, se tratará de un fichero secuenciales en el que se colocan datos. Si se pone INPUT, en cambio, se tratará de un fichero a partir del cual se pueden hacer entrar datos.
Si no se incluye el FOR <tratamiento>, entonces se trata de n fichero secuenciales de acceso directo (random file).
En combinación con OUTPUT e INPUT puede utilizarse APPEND. En ese caso se trata de ampliar un fichero secuencial.
La tabla siguiente nos enseña un resumen de las posibilidades.

<DEV>	*OUTPUT*	*INPUT*	APPEND	*Exclusión de FOR <tratamiento>*
CRT	*			
GRP	*			
LPT	*			
CAS	*	*		
A hasta F	*	*	*	*
COM	*	*		*

Con la expresión AS [#] se le asigna un número al fichero; este número se utiliza en las intrucciones INPUT#, PRINT#, y demás#. Concluimos finalmente recordando que si se utiliza la palabra <dev>, que este aparato realmente tiene que estar conectado. De lo contrario el ordenador dará una comunicacion de error.
Categoría: instrucción
Ejemplos: Vease CLOSE, GET, INPUT#, y LINE INPUT#

OUT <número de portál, expresión>
El dato de indicado por <expresión> es enviado al portál indicado por <número de portál>.
Ambas indicaciones deben estar entre 0 y 255.
Categoría: instrucción
Ejemplo: `10 OUT &HA8,INP(&HA8)`

PAD(<X>)
Con esta instrucción especial podemos investigar la situación de una tableta gráfica.

X	*se utiliza con*
0 hasta 3	tableta gráfica, enganchado al conector 1 de la palanca de mandos para juego
4 hasta 7	tableta gráfica, enganchado al conector 2 de la palanca de mandos para juego

Con la tableta gráfica se usa la tabla siguiente:

X	*significado del valor de la función*
0 o 4	determina si se ha tocado el tablero; 0 (no tocar) o −1 (tocar)
1 o 5	coordenada X del lugar tocado: PAD (0) o PAD (4) debe estar −1
2 o 6	coordenada Y del lugar tocado: PAD (0) o PAD (4) debe estar −1
3 o 7	determina si se ha pulsado el conmutador (0=no y −1=sí)

Categoría: función
Ejemplo:

```
10 SCREEN 2
20 AA=0
30 IF PAD(0)=0 THEN 20
40 X=PAD(1):Y=PAD(2)
50 IF AA=0 THEN PSET(X,Y) ELSE LINE-(X,Y)
60 AA=1
70 GOTO 30
```

PAINT [STEP](<X,Y>)[,[<color superficie>][,<color borde>]]

Indica que debe ser coloreado un espacio determinado. El sistema de coordenadas se desplaza por medio de STEP. Con x e y indicamos un punto. La superficie dentro de la cual cae ese punto, será coloreada. Si la línea del borde no es totalmente continua, será coloreada. Si la línea del borde no es totalmente continua, será coloreada toda la pantalla.
En los MODOS SCREEN 2 y 3 tienen que ser iguales <color superficie> y <color bordes>. En ese caso no se puede especificar el <color bordes>. En el MODO SCREEN 3 sí puede haber diferencias entre estas especificaciones de colores.
En los MODOS SCREEN 2 y 3, los números de colores deben estar entre 0 y 15.

Categoría: instrucción

Ejemplo:

```
10 SCREEN 3:COLOR 15,4,4
20 CIRCLE (80,80),40,8
30 PAINT (80,80),2,8
40 GOTO 40
```

PDL(<X>)

Da la posición de la palanca. Estando <X> entre 0 y 96. El valor de la functión varía de 0 a 255. En <X> = 1, 3, 5, 7, 9 o 11, el ordenador da por supuesto que el mando ha sido conectado por medio del conector 1. En <X> = 2, 4, 6, 8, 10 o 12, se supone la conexión 2.

Categoría: función

Ejemplo:

```
10 PRINT PDL(1):GOTO 10
```

PEEK(<X>)

Da el contenido de la dirección <X> de memoria. <X> debe encontrarse entre -32768 y 65536, en forma de número decimal. Si <X> es un número negativo, se adopta el llamado complemento a 2; PEEK(-1)=PEEK(65536−1).

Categoría: función

Ejemplo:

```
PRINT PEEK(65535)
```

PLAY [<primer canal>] [,[<segundo canal>] [,<tercer canal>]]

Instrucción para producir sonido, según determinadas indicaciones. PLAY "CDE" hace oír, por ejemplo, las notas do, re, mi. Para indicar una nota (do, re, mi, fa, sol, la, si y, asimismo, C+ por do sostenido, etc.) pueden usarse letras y números que tienen significados especiales:

Código	*Significado*
On	Da la octava correspondiente, por ejemplo: PLAY"05CDE": significa, toca do, re y mi de la octava 5. Para n es válido: 1<=n<=8. Si no se especifica nada el ordenador adopta 04.
Ln	Indica la duración de una nota del modo siguiente:

duración	*código*
4 tiempos	L1
2 tiempos	L2
1 tiempo	L4
1/2 tiempo	L8
1/4 tiempo	L16
1/8 tiempo	L32
1/16 tiempo	L64

	Si no hay ninguna especificación ulterior el ordenador adopta L4.
Nn	Estando n entre 0 y 96: indica una nota con número n.
A a G	Indica notas. Si se indica, por ejemplo, A5 esto corresponderá con 05A, o sea, la A de la quinta octava.
Rn	Indica un compás de silencio del modo singuiente:

duración	*código*
4 tiempos	R1
2 tiempos	R2
1 tiempo	R4
1/2 tiempo	R8
1/4 tiempo	R16
1/8 tiempo	R32
1/16 tiempo	R64

Vn	Indica el volumen: n=15 corresponde al 'volumen máximo'. Si no se especifica nada el ordenador adopta V8.
•	Alarga la duración de una nota con un factor 1,5.
Sn	Indica el timbre (0<=n<=15). Si no hay ninguna especificación ulterior el ordenador adopta S1.
Tn	Indica el movimiento. El valor de n determina la cantidad de negras por minuto. n puede caer entre 32 y 255. El valor estándar es 120.
Mn	Indica la medida en que varía el timbre. Sin especificación ulterior adoptará el ordenador M255 (1<=M<=65535).

Categoría: instrucción
Ejemplo: `PLAY "CDECDEEFGC"`

PLAY(<X>)

Da información sobre los canales de sonido utilizados: 0=todos los canales, 1=canal 1, 2=canal 2, 3=canal 3.
Si el canal de sonido indicado está siendo utilizado, PLAY retornara un valor –1. En los otros casos PLAY retorna un valor 0.
Categoría: función
Ejemplo:

```
10 PRINT PLAY(0)
20 PLAY "CDEFG"
30 PRINT PLAY(0):END
```

POINT(<X>,<Y>)

Da un numero que indica el color de un punto en la pantalla con las coordenadas X e Y. Las posibilidades de X e Y dependen del MODO gráfico indicado (véase SCREEN).
Categoría: función
Ejemplo: `PRINT POINT(50,50)`

POKE <dirección, dato>
Coloca un dato en el dirección de memoria indicado.
La <dirección> puede variar entre -32768 y 65535. El <dato> está entre 0 y 255.
Categoría: instrucción
Ejemplo: `POKE 5000,100`

POS(0)
Da la posición del cursor en la línea. La posición más a la izquierda es la posición 0. Este o es sólo un argumento imaginario y no tiene significado.
Categoría: función
Ejemplo: `10 SCREEN 0:LOCATE 10,20:PRINT POS(0)`

PRESET [STEP](<x,y>)[,<color>]
Coloca o borra un punto (x,y) en la pantalla gráfica. Si se ha expresado un color, el efecto de PRESET es el mismo que el de PSET. Sin esta especificación será borrado un punto que ya había sido indicado antes. Por medio de STEP puede desplazarsse, eventualmente, el sistema de coordenadas.
Categoría: instruccion
Ejemplo:

```
10 SCREEN 2
20 FOR K=1 TO 100:PRESET(K,K),1:NEXT
30 FOR K=1 TO 50:PRESET(K,K):NEXT
40 GOTO 40
```

PRINT [<expresión>[,o;][<expresión> <,o;>...]]
Reproduce en la pantalla el valor de variables, expresiones numéricas o literales. Las literales tienen que ir entre comillas (").
La posición, en que se colocan los datos en la pantalla, es determinada por el signo de separación. Si este signo es un punto y coma (;) o un espacio, entonces los datos serán colocados unos a continuación de los otros. BASIC divide una línea de zonas de 14 posiciones. Si el signo de separación entre dos datos es una coma (,) el segundo dato será colocado en la zona siguiente.
Si se cierra la instrucción PRINT con una coma (,) o un punto y coma (;), entonces serán colocados en la misma línea los datos de la instrucción PRINT que venga a continuación, y si no en la línea siguiente. El término PRINT puede ser indicado también por medio de un signo de interrogación, por ejemplo:

```
10 ? "RESULTADO";A
```

Categoría: instrucción
Ejemplo: `10 A=2:PRINT "MSX-";A`

PRINT USING "<código de notación>";<expresión>[;<expresión>...]
Esta instrucción constituye una ampliación de la instrucción PRINT en la que, con ayuda del <código de notación>, puede uno mismo inidicar en qué forma quiere que sean reproducidos los datos. Las <expresiones> tienen que ir separadas por un punto y coma (;).

En la reprodución de literales puede elegirse entre uno de los tres <códigos de notación> siguientes:
! Solo se reproduce el primer signo de escritura del líteral

\n espacios Ahora se reproducirán los dos primeros signos del líteral +n.

& Este signo indica que, no se imprime este signo, sino el contenido de la variable literal mencionada.

Para la reproducción de valores numéricos puede escogerse entre los <códigos de notación> especiales que se dan a continuación:

\# Indica las posiciones que hay antes y después de la coma. Las cantidades demasiado largas se redondean. Las cantidades demasiado pequeñas son adaptadas por medio de ceros y espacios. Si la cantidad es demasiado grande para el <código de notación> especificado, se colocará el signo de tanto por ciento (%) delante de la cantidad.
Ejemplos:

```
PRINT USING "##.##;1.2345
1.23
PRINT USING "##.##";99.996
%100.00
```

\+ Un signo más delante o detrás del <código de notación) hace que sea reproducido un signo más o menos delante o detrás de la cantidad.

– Un signo menos detrás del <código de notación> hace que aparezca un signo menos detrás de los números negativos.

** Un asterisco doble al principio del <código de notación> hace que sean reproducidos asteriscos en los espacios que eventualmente precedan a la cantidad. Además, estos dos asteriscos cuentan como dos posiciones extra.

$$ Un signo $ doble al principio del <código de notación> hace que sea reproducido el signo $ delante de la cantidad. La notación científica (véase más adelante) no puede ser utilizada en combinación con $$.

**$ Da una combinación de los dos efectos anteriores.

, Una coma a la izquierda del punto decimal hace que sea reproducida una coma cada tres posiciones.
Ejemplo:

```
PRINT USING "####,.##;1234.5
1,234.50
```

Una coma al final del <código de notación> es reproducida tal como está.
Ejemplo:

```
PRINT USING "####.##,";1234.5
1234.50,
```

^^^^ Colocando cuatro flechas al final del <código de notación>, se obtendrá la reproducción de la cantidad en notación científica.
Ejemplo:

```
PRINT USING "##.##^^^^";123.45
1.23E+0.2
```

Texto El texto que se coloque delante o detrás del <código de notación) será reproducido tal cual está, siempre que vaya separado del <código de notación) por un espacio.
Ejemplo:

```
PRINT USING "SON ##.## PESETAS";12.34
SON 12.34 PESETAS
```

Categoría: instrucción
Ejemplo: Véase lo anterior

PRINT # y PRINT # USING
La frase completa reza:

PRINT #<número de fichero>, [USING"<código de notación>";]<expresión> [;<expresión>...]

Sirve para enviar datos por escrito a un fichero. <El número de fichero> es el número que se ha asignado al fichero por medio de OPEN. Como <códigos de notación> son válidas las mismas posibilidades que han sido descritas en la instrucción PRINT USING. Las expresiones son enviadas unas detrás de las otras al fichero, debiendo utilizarse el signo de punto y coma (;) para separarlas. Si se utilizan comas entonces, los espacios que quedan agregados para la reproducción en la pantalla, serán enviados también al fichero. Las literales se colocan también unos a continuación de los otros en el fichero y, a causa de eso, ya no son reconocibles como literales independientes. Suponga, por ejemplo, que A$=JUAN y B$=PEDRO, la instrucción

```
PRINT #1,A$;B$
```

enviará al fichero JUANPEDRO como resultado. Eso sólo puede ocurrir cuando se hace salir como un solo literal. Este problema puede evitarse intercalando uno mismo signos de separación. Por ejemplo

```
PRINT #1, A$;",";B$
```

da como resultado JUAN, PEDRO
Otra posibilidad es la de enviar dos literales al fichero en dos instrucciones:

```
PRINT #1,A$
PRINT #1,B$
```

Si no se cierra la instrucción PRINT con un punto y coma, será añadido automáticamente un RETURN al final del líteral. Este RETURN hace las veces de signo de separación. Cuando el literal mismo contiene una coma, será considerado en la salida como si fuesen dos literales. Por ejemplo:
A$=JUAN,PEDRO y la instrucción

```
PRINT #1,A$
```

Da como resultado JUAN, PEDRO que sa puede obtener despues de:

```
INPUT #1, A$, B$
```

dará como resultado A$=JUAN y B$=PEDRO. Este problema puede evitarse poniendo entre comillas el líteral en el fichero (véase también la instrucción INPUT). La instrucción será entonces:

```
PRINT #1,CHR$(34);A$;CHR$(34)
```

34 es el código ASCII para las comillas (").
Categoría: instrucción
Ejemplo: véase lo anterior

PSET[STEP](<X,Y>)[,<Z>]
Con esto se situa un punto en la pantalla gráfica. Las coordinadas del punto concuerdan con X,Y. Los valores que pueden tomar X e Y dependen del MODO gráfico escogido (véase SCREEN). Con la Z se indica el número del color. En los MODOS SCREEN 2 y 3, Z puede variar entre 0 y 15. Si no se dan especificaciones, el valor de defecto es 15.
Categoría: instrucción
Ejemplo:

```
10 SCREEN 2
20 FOR K=1 TO 100:PSET (K,K):NEXT
30 GOTO 30
```

PUT[#]<número de fichero>[,<número de registro>]
Con éste podemos enviar el registro, que se encuentra en la memoria intermedia del fichero indicado, al fichero. El número bajo el cual se introduce el registro será una unidad mayor que el del último registro metido o sacado (si no se ha indicado número de registro), o será igual al número indicado.
Véase también FIELD, GET, LSET, RSET y OPEN.
Categoriá: instrucción
Ejemplo:

```
 10 OPEN "EXPL.DAT" AS#1
 20 FIELD #1,2 AS A$,10 AS B$
 30 FOR K%=1 TO 10
 40 INPUT N%,S$
 50 LSET A$=MKI$(N%)
 60 RSET B$=S$
 70 PUT #1,K%
 80 NEXT
 90 CLOSE #1
100 END
```

PUT SPRITE <número de plano>[,[STEP](x,y)][,<color>][,<número de sprite>]
Coloca el sprite con el color y el número indicados en la posición (x,y)
<plano numero> es el numero de prioridad del sprite. O es la mayor prioridad. Si dos sprites se encuentran en la misma posicion en la pantalla, entonces sera visible el sprite de mayor prioridad.
Por medio de STEP puede desplazarse eventualmente el sistema de coordenadas. Para un sprite de 8×8 puede tomarse como <número de sprite> un número de los comprendidos entre 0 y 255. Para un sprite de 16×16 uno de los comprendidos entre 0 y 63.
En los MODOS SCREEN 2 y 3 solo se pueden exponer 4 sprites uno al lado del otro (en una línea).
Categoría: instruccion
Ejemplo:

```
10 CLS:COLOR,11,11:SCREEN 2
20 A$="":FOR K=1 TO 8:A$=A$+CHR$(16):NEXT
30 SPRITE$(1)=A$:PUT SPRITE 0,(40,40),1,1
40 GOTO 40
```

READ<variable>[,<variable>...]
READ se utiliza siempre en combinación con DATA y asigna los valores especificados por DATA a las <variables>indicades.
Categoría: instrucción
Ejemplo: vease DATA

REM <commentario>
Por medio de ésta es posible incorporar comentarios al programa. Toda la información que haya después de REM será ignorada por BASIC. REM puede ser abreviada por medio del símbolo '.
Categoría: instrucción
Ejemplo:
```
10 REM BEETHOVEN
20 PLAY "GR8GR8GR8L2D+"
```

RENUM [<nuevo número de línea>][,[<antérior número de línea>][,[<tamaño del paso>]]]
Sirve para renumerar las líneas <nuevo número de línea> indica desde qué línea empezar. Si éste no se indica, se empezerá con 10.
<anterior número de línea> indica desde qué línea hay que empezar con la nueva numeración. Si no se indica, se empezará con la primera línea.
El <tamaño del paso> es el número con el cual hay que aumentar constantemente el número de línea. El estandar es 10.
Categoría: comando
Ejemplo:
```
RENUM 50,10,20
```

RESTORE [<número de línea>]
Sirve para leer de nuevo desde el principio, con ayuda de READ, las listas de valores especificados por DATA. Véase también DATA. Eventualmente, con el <número de línea> puede mencionar a una línea correspondiente, con DATA.
Categoría: instrucción
Ejemplo:
```
10 READ A,B,C:PRINT A,B,C
20 RESTORE
30 READ D,E:PRINT D,E
40 DATA 5,6,7
```

RIGHT$(<X$>),<I>)
Da un literal constituido por los últimos signos <I>de<X$>
Categoría: función
Ejemplo:
```
PRINT RIGHT$("MSX",2)
```

RND(<X>)
Da un número aleatorio entre 0 y 1. La serie de números aleatorios que se genere dependerá del número con que se comience.

<X>=negativo	En X=negativo se comienza por el principio de la serie de números aleatorios;
<X>=positivo	X=positivo da el número que sigue en la serie;
<X>=0	X=0 da otra vez el último número aleatorio.
	Con RND(-TIME) obtenemos números distintos cada vez (compare con RANDOMIZE de otras versiones de BASIC).

Categoría: función
Ejemplo:
```
10 FOR K=1 TO 100:PRINT RND(1):NEXT
```

RSET
Véase LSET

RUN[<X>]
RUN"[<dev>:]<nombre del programa>"[,R]
Podemos empezar un programa con RUN. Cuando se especifica X significa que el programa presente en la memoria debe ser comenzado a partir de la línea número X. Si especificamos <dev> significa esto que el programa se encuentra presente en un medio externo. En lugar de <dev> podemos llenar: CAS, A hasta F y COM [<n>].
RUN surte ademas el efecto de que todos los ficheros abiertos se cierran, a no ser que R se especifique.
Categoría: comando
Ejemplo: `RUN`

SAVE"[<dev>:]<nombre del programa>"[,A]
Con esto se almacena un programa de la memoria en el medio indicado (véase LOAD). En lugar de <dev> podemos indicar lo siguiente: CAS, A hasta F y COM [<n>].
El nombre del programa es el mismo que debe ser utilizado con LOAD y MERGE al cargar nuevamente el mismo. Con A se indica que el programa debe ser almacenado en código ASCII.
Categoría: comando
Ejemplo: `SAVE "CAS:DATA"`

SCREEN[<X>[,<Y>[,<Z>[,<XX>[,<YY>]]]]]
Con la instruccion SCREEN se indica como debe ser utilizada la pantalla. Cuando se enciende el ordenador, se supone automaticamente SCREEN 0,0,1,1,0.
Las indicaciones con letras tienen el siguiente significado:

<X>	*MODA: texto o grafico*
0	MODO de textos 1 con WIDTH 40 (40×24)
1	MODO de textos 2 con WIDTH 32 (32×24)
2	MODO grafico 1: 256×192 motas
3	MODO grafico 2: 64×48 bloques de motas

Existen las siguientes posibilidades para indicar colores:

<X>	*numéros de colores*
0	2
1	2
2	16
3	16

Las siguientes instrucciones solo se pueden usar en el MODO grafico: CIRCLE, DRAW, LINE, PAINT, PSET, PRESET, ON SPRITE GOSUB, SPRITE ON/OFF/STOP, POINT y PUT SPRITE.

<Y>	*formato de los sprites*
0	8×8 sin aumento
1	8×8, con aumento (factor 2)
2	16×16, sin aumento
3	16×16 con aumento (factor 2)

<Z>	*sonido o ningún sonido al pulsar tecla*
0	no da ningún sonido (beep) al pulsar tecla
1	da un sonido al pulsar tecla

<XX>	*velocidad de entrada/salida de cassette*
1	1200 baudios
2	2400 baudios

<YY>	*tipo de impresora*
0	impresora MSX
1	otra

Categoría: instrucción
Ejemplo: véase LINE, PSET

SGN(<X>)
En <X> = positivo será SGN(<X>) = 1
En <X> = 0 será SGN(<X>) = 0
En <X> = negativo será SGN (<X>) = -1
Categoría: función
Ejemplo: `PRINT SGN(31)`

SIN(<X>)
Da el seno de <X>.<X> tiene que ser expresado en radianes.
Categoría: función
Ejemplo: `PRINT SIN(3.1415/12)`

SOUND<nombre de registro, cantidad>
Instrucción para crear un sonido determinado. Existen los siguientes convenios:

registro	*alcance*	*significado*
0	0-255	frecuencia canal A
1	0-15	frecuencia canal A
2	0-255	frecuencia canal B
3	0-15	frecuencia canal B
4	0-255	frecuencia canal C
5	0-15	frecuencia canal C
6	0-31	frecuencia ruido
7	0-63	canal de elección: tono y ruido
8	0-15	volumen canal A
9	0-15	volumen canal B
10	0-15	volumen canal C
11	0-225	frecuencia de variación de la onda sonora
12	0-255	frecuencia de variación de la onda sonora
13	0-14	elección la onda sonora

Categoría: instrucción
Ejemplo: `10 FOR K=1 TO 10:SOUND K,0:NEXT K`

SPACE$(<X>)
Da un literal compuesto de <X> espacios. <X> es redondeada en número entero y tiene que encontrarse entre 0 y 255.
Categoría: función
Ejemplo: `10 A$=SPACE$(20):PRINT A$;"A"`

SPC(<X>)
Da <X> espacios en la pantalla o la impresora. SPC puede utilizarse sólo dentro de una instrucción PRINT o LPRINT. <X> tiene que encontrarse entre 0 y 255.
Categoría: función
Ejemplo: `PRINT "MSX";SPC(3);"2"`

SPRITE ON
SPRITE OFF
SPRITE STOP
Con ON, OFF y STOP se indica si el ordenador está atento a la 'colisión' de sprites. Véase ON SPRITE GOSUB. Por medio de SPRITE STOP se indica que hay que mantener el 'estado de interrupción'. En este caso, sólo volverá a saltarse a la subrutina especificada por ON SPRITE GOSUB, cuando se vuelva a encontrar SPRITE ON.
Categoría: instrucción
Ejemplo:

```
 10 DATA 60,66,165,129,165,153,66,60
 20 DATA 60,126,219,255,255,219,102,60
 30 A$=""
 40 FOR I=1 TO 8
 50 READ A:A$=A$+CHR$(A)
 60 NEXT
 70 B$=""
 80 FOR I=1 TO 8
 90 READ A:B$=B$+CHR$(A)
100 NEXT
110 SCREEN 2,1:COLOR 15,4,1
120 ON SPRITE GOSUB 210
130 SPRITE$(0)=A$:SPRITE$(1)=B$
140 SPRITE ON
150 A=INT(RND(1)*256):B=INT(RND(1)*256)
160 FOR I=0 TO 191
170 PUT SPRITE 0,(A,I),1
180 PUT SPRITE 1,(B,191-I),15
190 NEXT:GOTO 140
200 SPRITE OFF
210 PLAY "L4CDEFEDCREFGAGFER"
220 PUT SPRITE 0,(0,208)
230 PUT SPRITE 1,(0,208)
240 I=191:RETURN
```

SPRITE$
Aquí se trata de una variable de sistema que se utiliza siempre de la forma siguiente: SPRITE$(<número>)=(<expresión literal>). El <número> indica el llamado número de sprite. Dependiendo de la definición del formato del sprite, según se ha dado por medio de la instrucción SCREEN, podrá ser el número, 63 como máximo o 255. Con ayuda de la

<expresión litéral> indicaremos la forma del sprite. Esto lo hacemos normalmente por medio de algunas funciones CHR$, por ejemplo:

```
SPRITE$(1)=CHR$(&H18) + CHR$(&H3C) + CHR$(&HFF) + CHR$(&H99) + CHR$ (&H99) +
          CHR$(&HFF)+CHR$(&HC3)+CHR$(&HFF)
```

Para un sprite de 8x8 tenemos necesidad de unas 8 funciones CHR$.
SPRITE$ se puede utilizar en las diferentes MODOS gráficas (véase además PUT SPRITE).
Categoría: variable del sistema
Ejemplo: Véase SPRITE ON/OFF/STOP

SQR(<X>)
Da la raiz de <X>.<X> tiene que ser mayor o igual que 0.
Categoría: función
Ejemplo: `PRINT SQR(4)`

STICK(<X>)
Da la posición del mando (1 o 2). Si se concede a X el valor 0, hay que fijarse en las teclas del cursor.
Los valores corresponden a las siguientes direcciones:
Categoría: función
Ejemplo: `PRINT STICK(0)`

STOP
Interrumpe la ejecución de un programa. Podremos continuarla con el comando CONT.
En cualquier parte del programa se puede poner STOP.
Categoría: instrucción
Ejemplo:

```
10 INPUT A
20 PRINT A:IF A=0 THEN STOP
30 GOTO 10
```

STOP ON
STOP OFF
STOP STOP
Regula el modo, al ocurrir una interrupción provocada por CTRL/STOP (véase ON STOP GOSUB). En STOP STOP es mantenido el estado de interrupción, es decir que sólo se saltará al la subrutina encontrada por ON STOP GOSUB, cuando vuelva a encontrase STOP ON.
Categoría: instrucción
Ejemplo:

```
10 ON STOP GOSUB 50
20 STOP ON
30 INPUT A$
40 IF A$="END" THEN STOP OFF:END ELSE GOTO 30
50 PRINT "TECLEA END (+ RETURN)":RETURN
```

STRIG(<X>)
Indica si se ha pulsado el botón de juego del mando (−1= sí y 0= no).

<X>	*significado*
0	barra espaciadora
1 o 3	joystick 1
2 o 4	joystick 2

Categoría: función
Ejemplo: `PRINT STRIG(0)`

STRIG (<X>) ON
STRIG (<X>) OFF
STRIG (<X>) STOP
Regula el modo, al ocurrir una interrupción provocada por el joystick o la barra espaciadora (véase ON STRIG GOSUB). En STRIG STOP es mantenido el estado de interrupción, es decir que sólo se saltará a la subrutina indicada por ON STRIG GOSUB, cuando vuelva a encontrarse la instrucción STRIG ON.
Categoría: instrucción
Ejemplo:
```
10 CLS:ON STRIG GOSUB 40
20 STRIG(0) ON
30 GOTO 30
40 LOCATE 5,5:PRINT "PULSE LA BARRA DE ESPACIO"
50 FOR I=1 TO 300:NEXT:LOCATE 5,5:PRINT SPC(25)
60 RETURN
```

STR$(<X>)
Da un litéral que representa el valor decimal de <X>.
Categoría: función
Ejemplo: `10 A$=STR$(10):PRINT A$`

STRING$(<I>,<J>) o STRING$(<I>,<X$>)
Da un litéral compuesto de <I> signos iguales que tienen <J> como código ASCII o corresponden con el primer signo de <X$>.
Categoría: función
Ejemplo: `PRINT STRING$(4,65)`

SWAP<variable>, <variable>
Cambia el valor de las dos variables.
Esta claro que estas variables tienen que ser de la misma naturaleza.
Categoría: instrucción
Ejemplo: `10 A=3:B=5:SWAP A,B:PRINT A,B`

TAB(<X>)
Sitúa el cursor en la posición <X> de la línea. Si el cursor ya ha sobrepasado la posición <X>, no ocurre nada.
<X> tiene que encontrarse entre 0 y 255. O es la posición más a la izquierda. TAB puede utilizarse sólo en instrucciones PRINT o LPRINT.
Categoría: función
Ejemplo: `PRINT TAB(12);"*"`

TAN(<X>)
Da la tangente de <X>.<X> tiene que ser expresada en radianes.
Categoría: función
Ejemplo: `PRINT TAN(3.1415/8)`

TIME
La variable del sistema TIME varía 50 veces por segundo. Esta variable podemos utilizarla, concretamente, para obtener números realmente aleatorios por medio de la función RND (véase ésta).
Categoría: variable del sistema
Ejemplo: `PRINT INT(RND(-TIME)*6+1)`

TROFF
TRON
Por medio del comando TRON se conecta el ordenador en un estado tal que durante la ejecución del programa se visualice asimismo el número de línea con el que está ocupado el ordenador.
Esto simplifica el hallazgo de errores. Este estado se desconecta por medio de TROFF.
Categoría: comando
Ejemplo:
```
10 FOR I=1 TO 3
20 PRINT I
30 NEXT
40 END
TRON
Ok
RUN
[10][20]1
[30][20]2
[30][20]3
[30][40]
Ok
TROFF
Ok
```

USR[<n>](<X>)
Sirve para señalar un programa en lenguaje máquina. <n> es el número que se ha asignado al programa en lenguaje máquina por medio de DEF USR. Si se omite <n> se adoptará USR 0.
<X> es un argumento que se trasmite al programa en lenguaje máquina.
El parámetro X se pasa de esta manera:
1. valores alfanuméricos: La dirección &HF663 tiene 3 como valor. Las direcciones &HF7F8 y &Hf7F9 indican indirectamente dónde se encuentran estos valores. Este valor se encuentra almacenado en tres bytes: byte 1 indica la longitud, y byte 2 y 3 indican la dirección, del valor alfanumerico en la memoria.
2. valores enteros: la dirección &HF663 tiene 2 como valor. Las direcciones &HF7F6 hasta &HF7F9 contienen el valor entero.
3. Valores de precisión sencilla: la dirección &HF663 contiene un 4. Las direcciones &HF7F6 hasta &HF7F9 contienen el valor de precisión sencilla.
4. doble precisión: La dirección &HF663 contiene un 8. Las direcciones &HF7F6 hasta &HF7FD contienen el valor de doble precisión.

Los valores de retorno que la subrutina pasa al MSX-BASIC son almacenados por el código de maquina en las direcciones anteriores. CLEAR nos permite resevar el espacio necesario.

Categoría: función

Ejemplo:
```
 10 CLEAR 200,&HEFFF
 20 AB=&HF000
 30 FOR I=AB TO AB+9
 40 READ A$:A=VAL("&H"+A$)
 50 POKE I,A
 60 NEXT I
 70 DEFUSR=&HF000
 80 INPUT "TECLEA UN NUMERO ENTERO";A%
 90 PRINT "NUMERO ENTERO = ";A%
100 R=USR(A%)
110 PRINT "EL RESULTADO ES UN ENTERO MAS 1";R
120 END
130 DATA 23,23,4E,23,46,03,70,2B,71,C9
```

VAL(<X$>)

Indica el valor numérico de <X$>. Si el primer signo de <X$> no es +, -, & o una cifra, VAL será igual a 0.

Categoría: función

Ejemplo: `PRINT VAL("10")`

VARPTR(<variable>)

VARPTR(#<X>)

Da la dirección del primer valor que forma parte de la <variable>, o el primer byte del bloque guía de fichero.

Tanto <X> como la <variable> deben tener un valor antes de que se pueda acudir a la función.

Categoría: función

Ejemplo:
```
10 A=10:B=VARPTR(A)
20 IF B<0 THEN B=B+65536
30 C$="0000"+HEX$(B)
40 PRINT RIGHT$ (C$,4):END
```

VDP(<X>)

Contiene el valor de los registros VDP. X puede variar entre 0 y 8.

El registro 8 solo se puede leer, nunca escribir.

Categoría: variable del sistema

Ejemplo:
```
10 FOR I=0 TO 8
20 A=VDP(I):B$="00000000"+BIN$(A)
30 PRINT RIGHT$(B$,8)
40 NEXT
```

VPEEK(<X>)

Da el valor de la dirección <X> de la memoria de video. X puede variar entre 0 y 16383.

Categoría: función

Ejemplo:
```
10 A=VPEEK(0)
20 A$="00"+HEX$(A)
30 PRINT RIGHT$(A$,2)
```

VPOKE<X, dato>
Igual que POKE, pero aqui la <X> se refiere a una dirección de la sección de vídeo de la memoria RAM. <X> puede variar entre 0 y 16383 y el dato entre 0 y 255.
Categoría: instrucción
Ejemplo: `10 VPOKE(0),(VPEEK(0))`

WAIT<número de portal, expresión 1>[,<expresión 2>]
Concierne a la introducción de datos a través de la portal I/O con indicación de número de portal.
El valor introducido es cambinado con <expresión> a través de la llamada condición exclusiva OR. Este resultado se combina con <expresión 2> a través de la operación AND. Si el resultado de esto es 0, se proseguirá la entrada y si no es así, el ordenador continuará el programa en la instrucción siguiente. Si se omite la <expresión 2>, el ordenador adopta el valor de 0.
Categoría: instrucción
Ejemplo: `WAIT &HA8,240`

WIDTH (<número de columnas>)
Fija el número de columnas de la pantalla de texto.
En el MODO SCREEN 0 se pueden instalar de 1 hasta 40 columnas. En el MODO SCREEN 1 se pueden instalar de 1 hasta 32 columnas (véase SCREEN).
Hacemos notar lo siguiente. Al encender el aparato, se instala una imagen de acuerdo a el MODO SCREEN 0, de 37 columnas. Si pasamos a el MODO SCREEN 1, entonces la imagen consistirá de 29 posiciones.
Categoría: instrucción
Ejemplo: `SCREEN 0:WIDTH 40`

INDICE DE PALABRAS CLAVE

¿Qué significan las siglas MSX? ¿Por qué están destinados los ordenadores MSX a jugar el papel protagonista? ¿Qué puede hacerse con el MSX-BASIC? En este libro de cómoda lectura se encontrarán las respuestas a esas preguntas y a muchas más. Ha sido escrito partiendo de considerar al lector como un 'novato' en el mundo informático, y de revelarle placentera y sistemáticamente los fascinantes secretos de la programación en MSX-BASIC y la operatoría de periféricos como impresoras, lecto-grabadoras de cinta y ductoras de disco.

PHILIPS

PHILIPS

8622 541 00003